

Search and Planning under Incomplete Information
A Study using Bridge Card Play

Ian Frank



Ph.D.
The University of Edinburgh
1996

Abstract

This thesis investigates problem-solving in domains featuring incomplete information and multiple agents with opposing goals. In particular, we describe FINESSE — a system that forms plans for the problem of declarer play in the game of Bridge.

We begin by examining the problem of search. We formalise a best defence model of incomplete information games in which equilibrium point strategies can be identified, and identify two specific problems that can affect algorithms in such domains. In Bridge, we show that the best defence model corresponds to the typical model analysed in expert texts, and examine search algorithms which overcome the problems we have identified.

Next, we look at how planning algorithms can be made to cope with the difficulties of such domains. This calls for the development of new techniques for representing uncertainty and actions with disjunctive effects, for coping with an opposition, and for reasoning about compound actions. We tackle these problems with an architecture that identifies and resolves conflicts on the basis of probabilistic reasoning about resources. The prime contribution of this architecture is that it can construct plans which succeed under a large number of possible world states without having to consider each world state separately. By casting our algorithm within the classical plan-space planning paradigm we show how our techniques differ from, yet may be incorporated into, existing planning architectures.

The original motivation for this work came from the field of mathematical reasoning. In Edinburgh, a technique known as proof-planning has been developed to control the search for the proofs of mathematical theorems. The defining feature of this paradigm is that it restricts the available options at any stage of planning to a pre-determined set of possibilities. By applying proof-planning techniques to the domain of Bridge (although currently only in the sub-problem of play without a trump suit), FINESSE becomes capable of constructing plans that look ahead right to the end of the tree of possible actions. Against human players it is therefore less prone to the short-sightedness that afflicts the traditional look-ahead approach. This allows it to solve problems that are beyond the range of previous systems.

We also describe two useful side-effects of formalising the way in which Bridge can be played. Firstly, the high-level nature of the objects in FINESSE's plans, combined with the utilisation of a qualitative uncertainty representation language, gives the system the ability to produce meaningful textual explanations for its actions. Secondly, by distilling the large diversity of card-play examples found in the Bridge literature into a manageable number of concrete and distinct manoeuvres we arrive at a deeper understanding of the game itself.

Acknowledgements

Doing this PhD was hard! No, really, *really* hard! I couldn't possibly have got through without help from countless people. I'd like to thank them all in a great big list, but that would take too long. Here's a little list instead.

Thanks to:

- The Three Wise Men — Alan Bundy, David Basin, and Brian Drabble — who were my supervisors for this research. For the ideas, the support, the encouragement, the brainstorming sessions, for telling me when I was *too* incoherent, and for finally getting me to submit. Thank you. Thank you. Thank you.
- My friends. For just telling me I was too incoherent.
- My family. Of course. A special mention to My Dad, though, who (voluntarily) saved my thesis from the fate of never being read by anyone except academics.

This research was supported by an SERC/EPSRC Advanced Studentship to the author, award number 92314771. It was also made possible by the rolling funding BC/DAAD ARC grant (Project number 438) of the Mathematical Reasoning Group at Edinburgh University.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Ian Frank
Edinburgh
July 7, 1996

Publications

Portions of this thesis have already appeared in print, have been submitted for publication, or have been made publicly available:

- The main body of Chapter 6 has been submitted to the journal *Artificial Intelligence*, as a paper titled 'Search in Games with Incomplete Information: A Case Study using Bridge Card Play', co-authored with David Basin.
- Portions of Chapter 5 have appeared in [Frank *et al* 92].
- Portions of Chapter 5 also draw on [Frank 91].
- Appendix A was made available on the World-Wide Web at the URL <http://www.dai.ed.ac.uk/students/ianf/Research/> from where it has been downloaded over 450 times between May 1995 and May 1996.

Contents

Abstract	ii
Acknowledgements	iii
Declaration	iv
Publications	v
List of Figures	xiv
1 Introduction	1
1.1 Motivation	2
1.2 Aims	2
1.3 Achievements	3
1.4 Bridge	6
1.4.1 The Basic Rules	6
1.4.2 Who Wants To Be a Millionaire?	7
1.5 The Rest of This Thesis	8
2 A Good Deal of Bridge Literature	11
2.1 Computer Bidders	12
2.1.1 The Rule-based Approach	12
2.1.2 Interpreting Other Players' Bids	14
2.1.3 Incorporating Look-ahead	15
2.1.4 COBRA	16
2.2 Computer Defender/Declarers	18

2.2.1	Rule-based Systems	19
2.2.2	Tactics	20
2.2.3	Tackling Card Combinations	25
2.2.4	Double-dummy Programs	26
2.3	Miscellaneous Bridge Research	26
2.4	Commercial Bridge Programs	27
2.5	Comparison with Other Games	28
2.6	Summary	31
3	Planning Literature	33
3.1	Plan-space and State-space	33
3.1.1	Plan Representation	34
3.1.2	Search Techniques	34
3.2	Planning Systems	36
3.3	Refinement Search	39
3.3.1	Background	39
3.3.2	Basic Formalisation	40
3.3.3	Brief Discussion	40
3.4	Plan-space Planning as Refinement Search	42
3.4.1	Partial Plan Representation	42
3.4.2	Auxiliary Constraints	44
3.4.3	A General Plan-space Planning Algorithm	46
3.4.4	Applying the Framework to Classical Planning Systems	50
3.5	Summary	52
4	The Bridge Search Space Size	53
4.1	Preliminary Estimates	54
4.2	Shape	55
4.3	Tightening the Bounds	57
4.3.1	Factoring in the Number of Possible Deals	57
4.3.2	The Number of Possible Shapes of a Single Hand	59

4.3.3	The Number of Possible Shapes of The Unseen Hands	60
4.3.4	The Incomplete Information Search Space in Bridge	62
4.4	Double-dummy Bridge	66
4.4.1	The Expected Number of Legal Play Sequences	66
4.4.2	Interpretation	68
4.5	Summary	69
5	Proof-planning:	
	Solving Independent Goals using Tactics and Methods	70
5.1	Proof-Planning	71
5.2	Bridge Tactics	74
5.2.1	Cashing	74
5.2.2	Finessing	75
5.2.3	Ducking	80
5.2.4	Top Sequences	81
5.2.5	Summary of the Tactic Set	82
5.3	Representing the Defenders' Plays	83
5.3.1	Card Sequences	84
5.3.2	Critical Cards	85
5.4	Methods	86
5.5	FINESSE's Planning Algorithm	89
5.5.1	A Planning Example	90
5.6	Interface Issues	92
5.6.1	Tracing the Planner	94
5.7	Searching with Histories	96
5.7.1	General analysis	98
5.7.2	Performance	99
5.7.3	Memory Management	102
5.8	Summary	103
6	Search in Games with Incomplete Information	104
6.1	Introduction	105

6.2	Game Theory Background	107
6.2.1	The Extensive and the Normal Forms of 2-player Games	107
6.2.2	Minorant and Majorant Games: The Minimax Theorem	110
6.2.3	Pure and Mixed Strategies	112
6.2.4	Preliminarity and Anteriority	112
6.3	Equilibrium Points in Bridge	114
6.3.1	Bridge as a Game of Incomplete Information	114
6.3.2	The Best Defence Model of an Incomplete Information Game . .	116
6.3.3	Solving the Best Defence Model	118
6.4	Exhaustive Strategy Minimisation	118
6.4.1	The Algorithm	119
6.4.2	An Example	120
6.4.3	Comparison with Standard Minimizing	121
6.4.4	Possible Worlds	122
6.4.5	The Complexity of Exhaustive Strategy Minimisation	123
6.5	Bridge Architectures Based on Standard Minimizing	126
6.6	Repeated Minimizing Fails: Strategy Fusion	128
6.6.1	A Bridge Example	132
6.6.2	Some Practical Consequences	134
6.7	Non-locality (Repeated Minimizing Fails Again)	137
6.7.1	A Bridge Example	141
6.8	Summary	144
7	Identifying The Best Strategy: Tackling Non-locality	145
7.1	Representing Information Qualitatively	146
7.1.1	A One-pass Approach	146
7.1.2	Discussion	148
7.2	Parameterised Local Evaluation Functions	150
7.2.1	Doublethink	150
7.2.2	Oddthink	153
7.2.3	Payoff Reduction	155

7.3	Application to Bridge: The Interpreter Algorithm	161
7.3.1	Action at Leaf nodes	162
7.3.2	Action at Internal Nodes	162
7.3.3	Lines of Play	164
7.4	Representing Uncertainty in Bridge	165
7.4.1	Binary Strings	165
7.4.2	<i>C</i> -conjunctions	167
7.4.3	Redundancy	168
7.4.4	Subsumption	173
7.4.5	Tidying up	174
7.4.6	Generating Probabilities	174
7.5	Coping with Non-locality in Bridge	181
7.6	Summary	183
8	Interleaving Plans With Dependencies	185
8.1	The Problem	186
8.1.1	Combinatorial Explosion	187
8.1.2	Reasoning about Dependencies: Resources	189
8.1.3	Resources in Bridge: Leads and Entries	191
8.1.4	Coping with an Opposition	193
8.1.5	Problem Summary	195
8.2	Resource Profiles	196
8.2.1	Context-dependency	196
8.2.2	Considering all Possible Worlds	199
8.3	Refinement Search	203
8.3.1	Review	203
8.3.2	Interleaving as Refinement Search	205
8.4	Goal Selection and Establisher Selection	207
8.4.1	Finding a Solution vs Finding the Best Solution	207
8.4.2	Using Probabilities	207
8.4.3	Maintaining the Profiles	209

8.4.4	The Effects of Ordering	210
8.5	Auxiliary Constraints and Book-keeping	213
8.5.1	Partial Order Plan-space Representation	214
8.5.2	To Split or Not to Split?	216
8.5.3	The Effect of an Opposition	219
8.6	A Solution Constructor Function for Bridge	220
8.6.1	A Naïve Approach	221
8.6.2	Utilising an Uncertainty Representation Language	222
8.6.3	Redistributing the Search Burden	223
8.6.4	A State-based Search Alternative	224
8.6.5	Using a Network of Constraints to Guide Interleaving	225
8.7	Summary	231
9	Re-introducing Neglected Actions	233
9.1	The Simple Squeeze	234
9.2	Re-introducing Squeeze Plays Into the Interleaver	236
9.2.1	The Need For a Squeeze Tactic	237
9.2.2	A Tailor-made Tactic	237
9.2.3	Carrying Out The Interleaving	240
9.2.4	Constructing a General Method for Simple Squeezes	242
9.3	Performance & Discussion	242
9.4	Summary	244
10	Overall Architecture	246
10.1	A Simple Planning Loop	247
10.2	A Partial Ordering on Profiles	249
10.2.1	A Modified Planning Loop	252
10.3	Forming ‘Best-Case’ Profiles	253
10.3.1	A Simple Approach	254
10.3.2	Don’t Count Your Chickens...	255
10.3.3	Losers	257

10.4	An Improved Overall Architecture	259
10.5	Summary	259
11	Results	262
11.1	Single-suit Plans	262
11.1.1	Basic Performance	264
11.1.2	Comparison Against Bridge Encyclopedia	266
11.1.3	Discovery of Errors	268
11.1.4	Relaxing the Best Defence Model	271
11.2	Global Plans	272
11.3	Summary	280
12	Conclusions	282
12.1	Contributions	282
12.1.1	Search	283
12.1.2	Planning	283
12.1.3	Proof-planning	284
12.1.4	Bridge	285
12.2	Further Work	286
12.2.1	Play Module	286
12.2.2	Incorporating Information From the Bidding	286
12.2.3	A Bridge Tutoring System	287
12.2.4	Extension to Suit Play	288
12.2.5	Adding Inter-suit Methods and Tactics	288
12.2.6	Improving the Top-level Control Structure	288
12.2.7	Making Inferences Based on Opponents' Play Styles	289
12.2.8	Identifying More General Algorithms to Deal with Non-locality	289
12.2.9	Planning to Discover Information	289
12.2.10	Extension to Mixed Strategies	290
12.2.11	Choosing the Hardest Line of Play to Defend Against	290
12.2.12	Extension to Defender Play	291

A	An Overview of Commercial Computer Bridge Systems	292
A.1	BBC Bridge Companion	294
A.2	Bidding	294
A.3	Bridge Buff	295
A.4	Bridge Champion with Omar Sharif	296
A.5	Bridge for Windows	296
A.6	Bridge Master	297
A.7	Bridge Master	298
A.8	BridgeMate	298
A.9	Bridge Olympiad	299
A.10	Bridge Pal	300
A.11	Grand Slam	300
A.12	Meadowlark Bridge	301
A.13	Microbridge	302
A.14	Micro Bridge	303
A.15	Micro Bridge Companion	303
A.16	Omar Sharif's Bridge	305
A.17	Oxford Bridge 4	306
A.18	Positronic Bridge	306
A.19	Saitek Industries	308
B	Generating Explanations	310
B.1	Review	310
B.2	A Basic Solution	311
B.2.1	Pattern Matching	311
B.3	Further Improvements	314
B.3.1	Voids, Singletons and Doubletons	314
B.3.2	Symmetry Between <i>C</i> -conjunctions	315
B.3.3	Symmetry Within <i>C</i> -conjunctions	316
B.3.4	Intermediate Logical Forms	316
B.4	Technical Supplement	318

B.4.1	Reversing a Binary String	319
B.4.2	When a Defender Holds ‘at least n cards’	319
B.5	Summary	320
C	Further Examples	321
C.1	Summary of Performance	321
C.2	Individual Examples	323
D	User Manual	354
D.1	Implementation Overview	354
D.2	Initialising the System	355
D.3	Setting up Play Situations	355
D.4	Basic Interface	356
D.5	Interface Predicates to Planners	358
D.6	Interface Predicates to Bidding System	360
D.7	Notes on Implementation	361
D.8	Saved Games	362
E	Code	365
E.1	Ftp Instructions	365
E.1.1	Step by Step Instructions	365
E.2	Getting Started	366
	Glossary	367
	Bibliography	373
	Index	385

List of Figures

1.1	Thesis chapters describing the five main modules of FINESSE	4
2.1	Grabiner's test results for COBRA	18
2.2	An example Bridge deal from [Khemani 94, Page 292]	22
2.3	Setting up a squeeze in Khemani's example	23
2.4	An example Bridge deal from [Smith & Nau 93, Page 90]	24
2.5	CHINOOK's performance as a function of depth	31
3.1	A generic refinement search algorithm	41
3.2	A simplified partial plan (P_E) in a domain with three operators	43
3.3	How a ground operator sequence S can satisfy an IPC $\langle t_i, c, t_j \rangle$	45
3.4	The full representation of the partial plan P_E	46
3.5	Generalised plan-space planning algorithm	49
3.6	Characterisation of plan-space planners as instantiations of <i>Refine-Plan</i> .	51
4.1	Choices and resulting shapes for a single player	56
4.2	Number of ways to produce shapes for a single hand	59
4.3	Log-scaled graph of search space lower bound against frequency	63
4.4	Estimated search space complexities of common games	64
5.1	The state-space search in theorem-proving	72
5.2	The state-space search in Bridge card play	73
5.3	Card combinations in which declarer can cash winners	75
5.4	A simple card combination allowing a finesse	76
5.5	Card combinations allowing Type 2 finesses	77
5.6	A card combination allowing a Type 3 finesse	78

5.7	A more complicated finessing situation	79
5.8	A further example of a backward finesse	80
5.9	A card combination in which ducking may profit	80
5.10	A card combination illustrating ducking	81
5.11	Card combinations where declarer has high cards but no winners . . .	82
5.12	Example card combination with seven outstanding cards	83
5.13	The general form of a method/6 term	87
5.14	The general form of a method/3 term in the original FINESSE	88
5.15	Preconditions of the Type 1 finesse	88
5.16	A planning example	90
5.17	Key for Figure 5.16	91
5.18	Example graphical display of plan data structure	93
5.19	Example display of plan data structure showing pop-up windows	95
5.20	Example trace window showing menu options	96
5.21	Example showing repeated states on the first planning level	97
5.22	Four test problems for single-suit planning	100
5.23	Graphs of cpu time against history depth for four single-suit problems	101
6.1	The extensive form of a two player game with one chance move	109
6.2	The exhaustive strategy minimisation algorithm	119
6.3	MAX's expected payoffs when selecting strategy (1,1)	121
6.4	Adaptation of minimax to best defence, 2-player, extensive form games	122
6.5	Flattened tree of MIN and MAX moves in a domain with five worlds .	123
6.6	Complete binary tree of MIN and MAX moves	124
6.7	The minimax values, e_{ij} , of each move M_i under world w_j	127
6.8	A tree of MIN and MAX moves mis-analysed by exhaustive minimaxing	129
6.9	Two trees with the best initial move marked in bold	131
6.10	A Bridge situation requiring a guess over the best strategy	132
6.11	A Bridge example where strategy fusion obscures the best strategy . .	133
6.12	An example card combination used by Ginsberg	134
6.13	A simple tree with just one MAX choice	135

6.14	A Bridge deal illustrating discovery plays	136
6.15	Tree of MIN and MAX choices in a domain with five possible worlds .	137
6.16	General form of non-locality	139
6.17	Two trees with the best MAX strategy marked in bold	140
6.18	A Bridge card combination where one trick must be lost	142
6.19	Possible worlds under which each tactic will produce 2 tricks	142
6.20	A Bridge example giving rise to non-locality	143
7.1	Representing payoffs as ‘vectors’ of possible outcomes	146
7.2	Using vectors to describe the outcomes of possible strategies	147
7.3	Improving efficiency via the use of a simple propositional language . . .	149
7.4	Naïve form of a minimax algorithm on flattened trees	150
7.5	Simple example of a flattened tree	152
7.6	Carrying out doublethink	153
7.7	Simple example of how multiple levels of reasoning gives rise to loops .	156
7.8	Flattened tree example with one extra world added	157
7.9	The payoff reduction minimaxing algorithm	158
7.10	Applying the <i>reduce-payoff-mm</i> algorithm	159
7.11	Conducting minimaxing in each world concurrently	160
7.12	Modified form of the payoff reduction minimaxing algorithm	160
7.13	The use of subsumption to model minimisation of the possible payoff .	163
7.14	One player’s possible holdings from the sequence AKQ	166
7.15	Binary strings representing possible holdings from the sequence AKQ .	166
7.16	A simple card combination	168
7.17	East’s possible holdings of Kxxxxx	168
7.18	A composite tree describing three outstanding sequences	169
7.19	The paths allowed by the disjunction of two separate \mathcal{C} -conjunctions .	169
7.20	Transforming overlapping subtrees into new subtrees with no overlap .	171
7.21	Algorithm for removing redundancy in disjoint \mathcal{C} -conjunctions	172
7.22	The probabilities of the possible splits of 6 cards	177
7.23	The number of distributions described by (7.12)	178

7.24	Calculating the probability of the distributions described by (7.12) . . .	178
7.25	Algorithm to produce a shape-list for a single \mathcal{C} -conjunction	179
7.26	Trace of the shape-list algorithm	180
7.27	A Bridge example giving rise to non-locality, repeated	181
7.28	Second round situation (giving rise to non-locality)	182
7.29	Possible worlds under which second round tactics produce 2 tricks . . .	183
8.1	Linear lines of play	188
8.2	A simple cashing example	192
8.3	An almost solid suit	193
8.4	A possible deal containing an almost solid suit	194
8.5	A deal where the defenders must not be allowed to have the lead . . .	195
8.6	An almost solid suit, repeated	197
8.7	Resource consumption and production of tactics in a single line of play	198
8.8	Producing the resource analysis of a line of play bottom-up	199
8.9	Possible resources that can be (a) Required, and (b) Produced	200
8.10	A ducking example	201
8.11	Generalised plan-space planning algorithm (repeated)	204
8.12	A simplified refinement planning algorithm applied to interleaving . . .	206
8.13	A revised algorithm for resource-based interleaving	211
8.14	A plan set with resource requirements and an imposed ordering	211
8.15	A final algorithm for resource-based interleaving	213
8.16	Combining lines of play in a network	214
8.17	An example Bridge deal	215
8.18	Branching in a line of play	217
8.19	Another example of branching in a line of play	218
8.20	Lines of play fragments that can be interleaved	220
8.21	An interleaving strategy examining each possible world separately . . .	222
8.22	The use of uncertainty terms to improve search efficiency	223
8.23	Using a network to order plan actions	230
8.24	A simple Bridge example	231

9.1	A Bridge deal allowing a simple squeeze	235
9.2	Situation when squeeze is actually applied	236
9.3	Squeeze situation with defenders' cards swapped	237
9.4	A Bridge deal allowing a positional squeeze	238
9.5	Situation when positional squeeze is actually applied	239
9.6	A squeeze tactic	240
9.7	The interleaving of a squeeze play	241
10.1	Simple planning loop for an anytime architecture	248
10.2	Schematic representation of four plan profiles	250
10.3	Simple planning loop for an any-time architecture	252
10.4	An example Bridge deal	256
10.5	Planning loop modified for efficiency	260
10.6	The six modules of the FINESSE system	261
11.1	FINESSE's top-level interface	263
11.2	Simple card combination with no losers	264
11.3	Plan which discovers chance of winning extra trick	265
11.4	Card combination allowing finessing	265
11.5	Card combination offering a choice between finesses	266
11.6	A card combination from the Bridge Encyclopedia	267
11.7	A second card combination from the Bridge Encyclopedia	268
11.8	Card combination number 26 from the Bridge Encyclopedia	268
11.9	Distributions allowing four tricks to be taken, and their probabilities	269
11.10	A card combination analysed incorrectly in the Bridge Encyclopedia	269
11.11	Card combinations analysed incorrectly in an expert text	270
11.12	Combination where relaxing A-II can improve chance of success	271
11.13	Combination where relaxing A-I can improve chance of success	272
11.14	A deal with a certain plan for making the contract	273
11.15	Top portion of a solution constructor interleaving for Figure 11.14	274
11.16	An example Bridge deal, repeated	275

11.17	Top portion of a solution constructor interleaving for Figure 11.16 . .	276
11.18	A possible deal containing an almost solid suit	277
11.19	Top portion of a solution constructor interleaving for Figure 11.18 . .	278
11.20	Portion of solution constructor interleaving including a squeeze	279
11.21	An example deal from [Berlin 85]	280
A.1	Summary of commercial Bridge systems	293
B.1	Canned text for describing \mathcal{C} -conjunctions	312
B.2	Patterns matching canned text for quantity of cards held	313
B.3	Possible explanations for an outstanding sequence of Ace and King . .	313
B.4	Possible explanations for an outstanding sequence of 3 low cards	314
B.5	Scheme for reversing a binary string	319
C.1	Overall FINESSE performance on Bridge Encyclopedia problems	322
D.1	Database keys used to store state of game	362
D.2	Additional database keys used by FINESSE	363

A game may be as integral to a culture, as true an object of aesthetic appreciation, as admirable a product of human creativity as a folk art or a style of music; and, as such it is quite as worthy of study.

— MICHAEL DUMMETT
The game of Tarot

Chapter 1

Introduction

All reality is a game. Physics at its most fundamental, the very fabric of our universe, results directly from the interaction of certain fairly simple rules, and chance; the same description may be applied to the best, most elegant and both intellectually and aesthetically satisfying games.

— IAIN M BANKS
The Player of Games

[With] the kinds of problems that are completely specified ... in principle, if you work hard enough (*how* long may not be known), you can find the answer because all the information needed to solve the problem is right there with the statement of the problem. But in the real world the situation is seldom so pure and precise. Only some of the cards are faceup on the table.

— HEINZ PAGELS
The Dreams of Reason

This thesis is about incomplete information. In particular, it's about how solving a problem becomes more difficult when the problem is only partially specified. To make life even harder, we'll also assume that our problems include an opponent. These types of problems will be like the 'real world' of Pagels, in that they are not 'pure and precise', and also like the 'reality' of Banks, since we will use games as examples. Our particular example will be the game of Bridge, or more accurately Bridge card play. In Bridge, each player always has incomplete information because the game starts with the dealing of a deck of cards. Thus, the problems we tackle really will be real-world situations where 'only some of the cards are face-up on the table'.

1.1 Motivation

Originally, this work was motivated by wanting to apply the notion of *proof-planning* to new domains. Proof-planning is a technique originally developed at the University of Edinburgh to find proofs for mathematical theorems. However, mathematics has several features that make it considerably simpler than ‘real-life’ problems. For example, there’s no opposition, and no restriction on backtracking once the execution of a plan has begun. Also, there’s no incomplete information, so any action will always have a predictable outcome, and plans will never need to be revised in the light of new information. Further, since the only possible actions are proof steps carried out by the computer, there are never any problems with co-ordinating the actions of several agents.

All of these problems can be found in games, however, so adapting proof-planning techniques to a game-playing environment appealed as a natural intermediate step towards the goal of tackling real-world problems.

Why Bridge in particular? Well, one reason is that computer programs have yet to prove as successful in Bridge as they have in other games, such as chess, checkers, or backgammon. More compellingly, though, we recognised a basic similarity that Bridge shares with mathematical theorem-proving: that is, despite an incredibly large search space, human experts are still able to identify good solutions, and have built up a large variety of techniques for successful problem-solving. It was the success of proof-planning in declaratively capturing such human problem solving strategies in theorem-proving that was the main motivation for wanting to apply the same techniques to Bridge.

1.2 Aims

The main aim of this research was to build a system that utilised proof-planning ideas to control the search in the planning of Bridge card play. By producing a working system, we hoped to learn about planning and search in domains with incomplete information in general, and also about the game of the Bridge itself. We restricted ourselves to

the task of *declarer play*, where one player tries to play his cards to achieve a specific goal that is decided upon before the start of the play. Also, although we constructed an architecture general enough to deal with every possible aspect of declarer play, we mainly concentrated on the sub-problem of *No Trumps* play, where each suit is of equal value.

1.3 Achievements

The concrete result of this research was the FINESSE system. FINESSE is a body of Prolog code consisting of five main modules: a *pre-planner*, a *planner*, an *interpreter*, an *interleaver* and a *play* module. A simplified outline of the system architecture is shown in Figure 1.1, which also indicates the thesis chapters that describe each module.

Prior to actually initiating any planning, a small amount of pre-computation is carried out by the pre-planner in order to determine the significant missing high cards (the *critical cards*). The planner then combines this information with the original game state and uses proof-planning techniques to produce a (minimax) tree of possible actions in each suit. These trees are not immediately executable as plans, however, since they contain branching points at which choices between the available paths will have to be made. To facilitate this selection, the trees are analysed by the interpreter, which produces qualitative statements about the subdivisions of the outstanding cards that would lead to the ‘success’ or ‘failure’ of the *lines of play* contained in the plan. The interleaver then uses this information to select lines of play to interleave. This interleaving involves the posting of ordering constraints and also the introduction of *inter-suit* operators to represent actions which may have been neglected by the original decomposition into separate suits. Different combinations of lines of play are tried on backtracking until planning time is exhausted, when the best global plan so far (as determined by the *probability profiles*) is returned. The actual style of play can be varied by choosing between these profiles on the basis of different criteria.

We can categorise the lessons learned from this implementation into the following general areas:

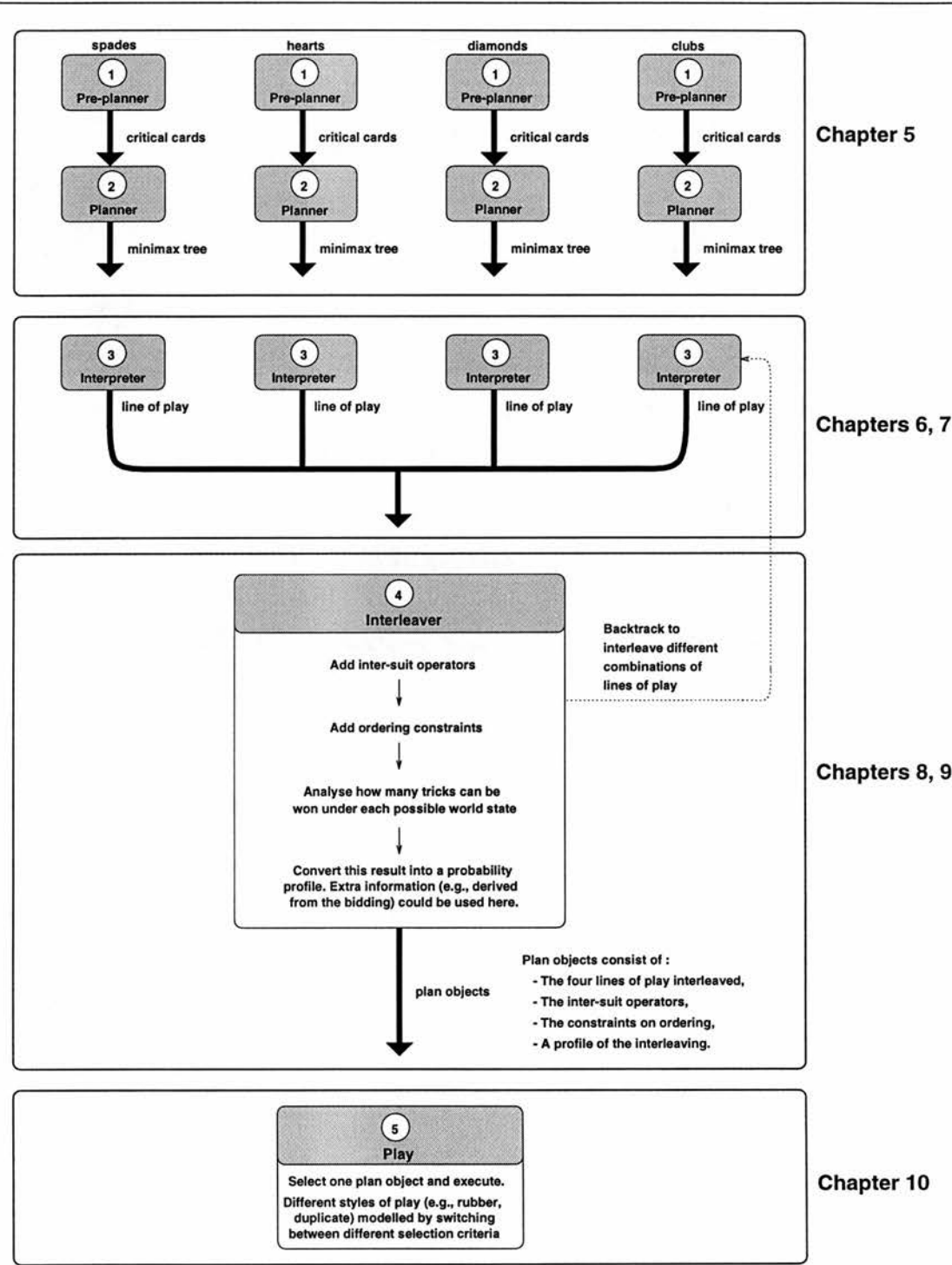


Figure 1.1: Thesis chapters describing the five main modules of FINESSE

Search

We show that adaptations of traditional algorithms are not sufficient to cope with domains with incomplete information, and formalise two problems that can occur in such domains. For the particular problem of Bridge card play we also demonstrate how these problems can be overcome by the use of domain-specific heuristics. (Chapter 6 and Chapter 7)

Planning

Under incomplete information there are many possible states of the world, and plans may only succeed under a small subset. However, typical classical planning algorithms function by introducing constraints until a plan is *guaranteed* to succeed. This will not always be possible in domains with incomplete information, and our architecture introduces new techniques for coping with this. By casting our algorithm within the classical plan-space planning paradigm we show how our techniques differ from, yet may be incorporated into, existing planning architectures. (Chapter 8)

Bridge

We produce new lower bounds on the size of the search space size involved in Bridge card play. These bounds are tighter than those developed by previous authors and allow us to make more informed speculations on the future of computer Bridge research. (Chapter 4)

In addition, the creation of FINESSE's declaratively encoded methods formalised knowledge which was only previously expressed in the form of examples in the Bridge literature. This is analogous to the way in which the study of proof-plans at Edinburgh has led towards a theory of inductive theorem proving. (Chapter 5)

The Proof-planning Paradigm

FINESSE itself proved to be a capable system, especially in single-suit problems, consist-

ently identifying a best plan that matched or was very close to the recommended lines of play found in the Bridge literature. In some cases, FINESSE's analysis even revealed shortcomings in the explanations presented by recognised authorities. These examples were generally situations in which authors had overlooked particular subdivisions of the outstanding cards that would lead to the success or failure of a line of play. Such 'discoveries' demonstrate the fallibility of humans, even when they are 'experts', in dealing with problems that involve combinatorial explosion. FINESSE's ability to cope with such problems can be directly attributed to use of the proof-planning paradigm, which results in a search space typically four or five orders of magnitude smaller than the space that would result from considering all the possible ways the cards could be played. This reduction makes analysis of the entire search space computationally feasible. (Chapter 5)

Moreover, FINESSE has given us valuable insight as to how proof-planning techniques can be extended to 'adversarial domains' that necessitate disjunctive choice based on probabilistic or uncertain information, and to domains which involve reasoning about interactions between plan steps. (Chapter 9)

1.4 Bridge

To provide an adequate background, we will present a brief introduction of the game of Bridge. To do the game justice, however, would require many pages, and is beyond the scope of this thesis. We therefore simply present a summary of the basic rules, and refer those readers interested in further details to one of the many excellent books on the subject (see the Bibliography for suggestions).

1.4.1 The Basic Rules

Bridge is a card game played with a deck containing 52 cards, comprised of 4 suits (spades ♠, hearts ♥, diamonds ♦, and clubs ♣) each containing the 13 cards Ace, King, Queen, Jack, 10, ..., 2 (we will sometimes abbreviate the first five of these to A, K, Q, J and T). The game begins with the chance move of shuffling the deck, and the cards are then dealt between four players, traditionally named North, South, East and

West. These players form two teams: North/South against East/West.

Card play starts when one player lays a card on the table, which all the other players then cover in turn (in a clockwise direction) with a card from their own hand. Each round of four cards is called a *trick*, and the winner of one trick becomes the first person to play a card on the succeeding round. Before this play begins there is an *auction* during which one team wins a *contract* to make a certain number of tricks. One member of the winning team becomes the *declarer* and the other is the *dummy*. The other team are the *defenders*. The defender to the left of the declarer starts the card play, and the dummy's cards are then laid face-up on the table from where they are played by the declarer, along with his own cards. For simplicity, our examples will usually assume that South is the declarer, so that North is the dummy and East and West are the defenders. For the purposes of this thesis, the only significant rules will be:

- The first player in a trick can freely choose which card to play from all those present in his hand.
- Subsequent players must *follow suit* by playing a card of the same suit as the one that started the trick, if they hold such a card — if they do not (*i.e.*, they are *void* in the suit), they can make a free choice from among the remaining cards in their hand.
- The winner of the trick is the player who plays the highest card (ranked by $A > K > Q > J > 10 > \dots > 2$) of the suit led. The only exception to this when there is a suit declared as the *trump suit*; if any trump cards are played, then the player playing the highest trump card is the winner. Playing a trump when you cannot follow suit is known as *ruffing*.

1.4.2 Who Wants To Be a Millionaire?

The poor performance of computer Bridge programs to date has prompted the former World Champion Zia Mahmood to offer one million pounds to the designers of a computer system capable of defeating him. Although it seems that he may now want to withdraw this bet (perhaps worried by the progress of research in other games like

chess and checkers), at the moment it still appears that his money should be completely safe. For example, Brent Manley, Bridge expert and reviewer of Bridge programs for the magazine of the American Contract Bridge League, has this to say about the problems facing would-be millionaires [Manley 94]:

It is generally accepted that writing a computer Bridge program that plays Bridge well is nearly impossible.

Tom Throop, one of the leading producers of commercial Bridge software, is also far from sanguine about the prospects of computer Bridge [Throop 83]:

The task of writing a Bridge-playing program that demonstrates a high level of intelligence is actually more difficult than writing a similarly intelligent chess program.

In a more optimistic vein, though, Christina Erskine, Bridge-playing editor of the magazine *PC Review*, writes [Erskine 92]:

One day, someone will program bridge to play more like an expert (and a very long, fascinating exercise in artificial intelligence it will be, too) and put all those chess programs in their place.

Like those that have gone before, we haven't succeed in producing an expert-level Bridge program. But we hope that we have shown that the task isn't quite the impossibility that Manley claims, and is also maybe not even as difficult as Throop suggests. And, of course, we'd like to think that we've definitely shown it to be the fascinating exercise in AI that Erskine envisaged. And maybe even as long...

1.5 The Rest of This Thesis

Here's how the thesis is laid out:

Chapter 1, *Introduction*. The current chapter. By now, you should have a good idea whether the rest of the thesis is for you or not.

Chapter 2, *A Good Deal of Bridge Literature*. Describes the previous work on computer Bridge.

Chapter 3, *Planning Literature*. Gives some necessary background on AI planning research.

Chapter 4, *The Bridge Search Space Size*. Just how hard is Bridge anyway? We establish some new lower bounds on the search space size.

Chapter 5, *Proof-planning: Solving Independent Goals using Tactics and Methods*. This is where the description of the FINESSE system begins. This, and each of the following chapters, describes a particular aspect of the overall architecture. Here, it's how to solve a problem by breaking it down into sub-problems and solving each of those in turn. Figure 1.1 on Page 4 has already shown which bits of the system are described in the other chapters.

Chapter 6, *Search in Games with Incomplete Information*. What algorithms can be used in this kind of game? What problems can occur?

Chapter 7, *Identifying the Best Strategy: Tackling Non-locality*. In fact, when there's incomplete information algorithms like minimaxing suffer from a problem we call non-locality. Here, we look at some work-arounds for this, including some heuristics specific to Bridge.

Chapter 8, *Interleaving Plans with Dependencies*. The plans here are plans for a single suit in Bridge; the dependencies arise because typically what you do in one suit affects what you can do in another. Putting them together isn't always easy.

Chapter 9, *Re-introducing Neglected Actions*. Just breaking a problem up and then putting together the plans for the sub-problems doesn't always solve your problem. Sometimes you actually have to consider more than one sub-problem at a time to come up with the best answer.

Chapter 10, *Overall Architecture*. This chapter pulls together the developments in the rest of the thesis and makes a coherent whole out of them.

Chapter 11, *Results*. We give examples of the system at work on examples taken from Bridge books or randomly generated.

Chapter 12, *Conclusions*. The conclusions. What would you expect?

Appendix A, *An Overview of Commercial Computer Bridge Systems*. Reviews of commercial systems that didn't fit into the Bridge literature review of Chapter 2 because the only information about them was about their level of play, and not *how* they did it. They're also at the end because there are too many of them to fit into the main text.

Appendix B, *Generating Explanations*. FINESSE can explain its decisions using plain English text. Here, we tell you how it's done, using plain English text.

Appendix C, *Further Examples*. A collection of examples from the *Bridge Encyclopedia* illustrating FINESSE's performance in detail.

Appendix D, *User Manual*. Gives you all the information you need to know to run the system.

Appendix E, *Code*. Gives you all the information your computer needs to know to run the system.

A *Glossary* and an *Index* are also included to help make sense of the thesis itself.

Chapter 2

A Good Deal of Bridge Literature

Rincewind looked blankly at Ysabell as words like ‘rebiddable suit’, ‘double finesse’ and ‘grand slam’ floated through the velvet.

‘Do you understand any of that?’ she asked.

‘Not a word,’ he said.

‘It sounds awfully complicated.’

On the other side of the door the heavy voice said: DID YOU SAY HUMANS PLAY THIS FOR FUN?

‘Some of them get to be very good at it, yes. I’m only an amateur, I’m afraid.’

BUT THEY ONLY LIVE EIGHTY OR NINETY YEARS!

— TERRY PRATCHETT
The Light Fantastic

Behaviour of the program is heuristic in nature. The program evaluates trick objectives, formulates goals, and executes the play of the declarer’s and the dummy’s cards. The program performs in accordance with heuristic principles I have always practised in tournament play and which I incorporated into the computer program.

— T. THROOP
Computer Bridge

Yes, humans play Bridge for fun. Like most things that are fun, though, it’s difficult to enjoy alone. And, since it can sometimes be hard to find others who share your taste in pastimes, ingenious designers have come up with computer programs to substitute for live participants. The three basic features to be found in such programs are:

- **Bidding.** The ability to bid for one or more of the players, according to the particular bidding system incorporated into the program (examples of bidding systems are *ACOL*, *Standard American*, *Schenken*, and *Kaplan-Sheinwold*).
- **Declarer Play.** The ability to play the declarer's and dummy's hands at a contract specified by the program or the user.
- **Defending.** The ability to play one or more of the defenders' hands in an attempt to defeat the declarer.

The automation of these tasks has been carried out by both academics and by producers of commercial software. We survey the academic research by reviewing the published papers and literature. For commercial products, on the other hand, there is usually scant information on their actual workings, with Throop's description above being rather typical of the level of detail available. For these systems, then, we will just present assessments of their playing strengths. These will be based on both our personal experiences with the programs and on reviews drawn from the specialist computer game and Bridge press.

2.1 Computer Bidders

We begin by examining the first broad category of computer Bridge programs — computer bidders. By the very nature of the bidding process, such programs are all basically rule-based systems. The area where they differ is in their approach to interpreting the bids made by the *other* players in an auction.

2.1.1 The Rule-based Approach

The first Bridge bidding program to appear in the literature is that of Carley [Carley 62]. This system is essentially intended as a card playing system, but it also incorporates a small bidding component, consisting of just four bidding rules with a total of 13 cases. Carley's own assessment of the bidding [Carley 62, Page 7] is:

The ability level is about that of a person who has played a dozen or so hands of bridge and has little interest in the game.

This level of bidding was acceptable for Carley's program, as it was primarily designed to cope with card play, and also because 'an extremely sophisticated bidding program' was already 'well under way' at Carley's institution. However, there is no record of this more sophisticated program in the literature, so the next Bridge bidder we find is that of Wasserman [Wasserman 70]. Like Carley's program, this is essentially a rule-based system — the difference being largely in the number of rules involved.

Wasserman divides his rules into a collection of *classes* which handle different types of bid, such as opening bids, responding bids or conventional bids. These classes are then organised by a set of *procedures* into *sequences* which specify the allowable orders in which the bids from each class may be used. Examples of these procedures are those that handle sequences starting with a bid of 1♣, or sequences that start with a 'weak two' bid. Wasserman views these procedures as reducing the search space size of the possible bidding sequences by identifying specific sequences to which attention is restricted.

Wasserman's system uses the standard Bridge bidding practice of assigning any hand a number of *points* based on its high cards, or the length of each suit. The strength of a hand measured by these points then helps to determine an opening bid, and this bid in turn determines a specific procedure. This procedure is then used for all subsequent bids of the partnership, determining the classes of rules to be used for each bid. However, there is a further significant addition. Before each bid is made, the bidder's hand is 're-evaluated' to take into account the history of the bidding sequence, considering such factors as awarding extra points on the basis of length in partner's bid suit(s) and adding/subtracting points based on shortness/length in his unbid suit(s). Wasserman describes the program's use of these factors as attempting 'to bid in the same manner as a human would, using the same decision-making criteria'. We will return to this issue in the following section.

Wasserman presents test results for his program that show it to achieve 'the level of human experts' in unopposed bidding sequences. These tests are based on bidding problems collected from the magazine *Bridge World*, involving actual deals in which each possible contract is awarded a number of points by the editors of the magazine.

Any partnership may then test their skill by bidding the hands together and summing their score. On a set of hands for which the editors of *Bridge World* considered a score above seventy to be outstanding, a score above sixty excellent, and a score above fifty to be above average, Wasserman's program scored 58.2. Further, Wasserman added extra bidding classes to his program to enable different bidding systems to be used (thus implementing the Standard American and the Kaplan-Sheinwold systems as well as the original Schenken), which he also tuned to produce a comparable performance.

In addition to normal partnership bidding Wasserman extended his program to handle competitive bidding by adding classes for bids such as take out doubles, penalty doubles and overcalling. In a 30 hand contest against two human Life Masters, split into two sessions, the program was victorious in the first session, defeated in the second, and narrowly beaten overall. Based on these results, Wasserman estimated that 'the program was slightly more skillful than the average duplicate Bridge player at competitive bidding'.

2.1.2 Interpreting Other Players' Bids

Wasserman points out that Bridge bidding presents 'many opportunities for the exercise of *judgement* and *imagination*, since no Bridge bidding system specifies action to be taken in all cases'. We saw above that Wasserman's program contains algorithms which deal with such considerations by adjusting the evaluation of any hand depending on the bidding history. We also saw that Wasserman justified this approach by appealing to parallels with human thought processes. Others have argued, however, that the 'human' approach more closely resembles reasoning with *explicit inferences* about the meanings of previous bids by the other players. For example, Stanier presents a discussion of the particular sequence of bids where North opens 1♣ and South responds with 1♠. Assuming the standard ACOL system, South's bid could simply be interpreted as showing at least 6 points and 4 or more cards in the spade suit. However, Stanier also presents the following deeper analysis [Stanier 75, Page 376]:

If we assume that South is making what he believes to be his best bid, then the bid of 1♠ describes his hand more precisely than any other. From this we may infer that he holds more spades than, say diamonds, as with longer diamonds, or with an equal number, he would have bid 1♦ (it being

a cheaper bid). Similarly, he cannot hold more than three clubs, or he would have bid 2♣. More information is available from the facts that he did not bid 1♥, 1No Trumps, etc.

The implication is that directly representing such inferences would reflect more closely the type of reasoning carried out by human players of the game. Also, the ability to make inferences in this form would be highly useful to a card playing component called upon to make or defend a contract. However, the performance of programs attempting to explicitly represent such information has yet to be proven in practice. For example, [MacLeod 91] describes a system which attempts to use the bids of any player to build a ‘picture’ of cards he holds (MacLeod’s system is also described in §A.13). Unfortunately, the representation used for capturing such information is not capable of coping with disjunction, and when any bid with more than one possible interpretation is made, information is therefore lost. This illustrates an important obstacle in the implementation of such systems: designing a representation language in which expressions describing *all* the possible inferences about each hand can be automatically generated. One technique for this which could be borrowed from mathematical reasoning is the use of a set of confluent rewrite rules to produce expressions in a normal form. However, we can already see from the above discussion that there are many possible types of inference that can be made about a single hand, including the number of cards in each suit, the maximum and minimum number of total points, the maximum and minimum length in each suit, and the relative lengths of each suit. Further, the position of particular high cards is often important, as is the breakdown of a hand’s overall points into the points held in each suit. Since most of these possible types of inference can interact, it becomes difficult to prove termination of any given re-write set. Such problems were encountered in the work of [Asher 93] and [Green 95].

2.1.3 Incorporating Look-ahead

An attempt to create a more general system capable of Bridge bidding is found in [Stanier 75]. Stanier’s aim was to ‘keep knowledge of Bridge peripheral to the program, and use a general core’. Stanier reasons that whilst bidding systems ‘provide useful guidelines for the bidding, they by no means make the process of bidding mechanical’, and he therefore argues that an element of planning should occur during the bidding

phase in order to produce realistic performance. Although this is a worthy aim, the problems involved are even larger than those in making inferences from single bids, and Stanier eventually concedes that ‘it was found necessary to put this [domain] knowledge in the core itself’.

Another system which proposes the use of look-ahead search over the space of possible bids is given in [Gambäck *et al* 91]. In this system, the rule-based portion is used simply to *suggest* possible bids for a player. These bids are then evaluated by conducting a look-ahead search; at each node, the valid inferences about each hand are made, a set of hands consistent with the constraints is generated, and a neural network is used to assess the chance of making the contract under each deal. The top-level control of this process is carried out by the further addition of a Bayesian planning system. Unfortunately, no test results are given.

2.1.4 Cobra

Judging purely by performance, the best bidding systems are those which do no look-ahead search and deal with the problem of inferencing using quantitative adjustments to hand evaluations rather than by explicitly representing inferences. The epitome of this approach is Lindelöf’s COBRA [Lindelöf 83].

In COBRA, each player’s hand is assigned a number of points as a ‘measure of the probable trick winning power of a hand’. At the beginning of the game, with no information available about the other hands, this evaluation is referred to as the *basic points count* (BPC) and is based on the assignment of points for each high card (A=4, K=3, Q=2, J=1), with extra points added and subtracted for considerations such as distribution, controls, shape and unprotected honours. The BPC ‘is chosen so that any opening promises 13+ points’ (except for bids by the third player). Its exact specification was produced by examining ‘several million hands’ which were randomly generated and assessed largely automatically.

During the auction, the points count of any player is adjusted and is then referred to as the *distributional points count* (DPC). The actual rules for this are not complicated, but take three pages of Lindelöf’s book to describe [Lindelöf 83, Pages 31–33]. Basically,

points are added or subtracted from the BPC for short suits, fit with partner, fit with the opponents, and other such considerations. These modifications are designed so that a joint value of 28 between two hands ‘gives the partnership a better than 50% chance of making ten tricks in their best trump suit’ with any additional trick requiring another 3 points. Again, these modification rules were tested by generating large numbers of random hands.

Lindelöf claims that COBRA’s bidding is of world expert standard, and provides evidence that in fact make this assessment sound rather modest. On a collection of 37 sets of ‘challenge’ hands from Bridge Magazine, taken from the period 1973 to 1979, COBRA scored an average of 89% and only once finished below the leading human pair. On another test, COBRA was used to replace the bidding of the human teams in the 1975 Bermuda Bowl final. Lindelöf claims that COBRA would have improved the North American score by 98 IMPs (International Match Points) or the Italian score by 69 IMPs — enough to translate the narrow Italian victory of 214–189 into a comfortable win either way.

An independent test of COBRA was carried out by David Grabiner [Grabiner 94]. Under similar criteria to that of Lindelöf, Grabiner used COBRA to replace the human players in the 1987 Bermuda Bowl and Venice Trophy. The results of this test are summarised in Figure 2.1, showing that the total swing produced by using COBRA would have been +74 over the 512 deals. Although this is still a positive result, it is much less than Lindelöf’s +167 in 192 boards. However, Grabiner attributes some of this discrepancy to the likelihood that Lindelöf’s book is not complete in some areas, and is also open to mis-interpretation.

Whatever the exact strength of COBRA it seems indisputable that it makes expert-level Bridge bidding a reality. The approach has been criticised, however, by [Gambäck *et al* 91] on the grounds that it will ‘only have a very limited ability to reason about its opponents bids’ and that this will be a ‘serious problem’, since ‘successful play requires that the bidding component is able to pass on the (probabilistic) information extracted from the bidding to the playing component’. This objection can be tackled by noting that a separate deduction system, such as that of [Quinlan 79] (see §2.3) could be used to produce these types of inferences. Also, it should be clarified that it is not the case

Tournament	Actual Result	Difference when replacing one team with COBRA	
Bermuda Bowl	US – GB 241 – 148	COBRA – GB +15	US – COBRA +43
Venice Trophy	US II – France 254 – 219	COBRA – France -2	US – COBRA +18

Figure 2.1: Grabiner's test results for COBRA

that COBRA is only a world-class bidder 'when it is allowed to bid undisturbed towards a slam' as [Gambäck *et al* 91] suggests. In fact, the test results summarised above were all produced under conditions of competitive bidding by all four players. COBRA has a full set of rules to cope with both defensive and pre-emptive bidding, and indeed the random hands on which COBRA was optimised necessarily allowed for bids by all four players, since a bidding system designed for uncontested auctions would have been heavily skewed towards slow development. The experimental evidence, then, is that the technique of calculating a numerical measure of each hand simplifies the intricacies of reasoning with qualitative information very effectively. One of the few problems overlooked by such an approach is the desirability of reaching the optimal contract whilst giving as little information about your own cards to the opposition. However, this may also be incorporated in future versions of the system.

2.2 Computer Defender/Declarers

The search space in a Bridge card play problem contains at least 9.91×10^{22} legal play sequences (see Chapter 4). The main pre-occupation of programmers attempting to automate card play has therefore been the effective control of the search through this space of possibilities. Below, we identify and discuss four techniques that have been used for this purpose.

2.2.1 Rule-based Systems

We have already mentioned the early research of [Carley 62] in the context of bidding. In fact, the rule-based operation of Carley's bidding system is almost identical in principle to that of the play program with which it is coupled; the issue of search is side-stepped completely by simply specifying a set of rules for actions to take in certain situations. Carley describes the resulting system as 'essentially a long list of rules of thumb'. An example of one of these is:

Defender, can follow suit, third card to
trick, second card high, probability the trick → play lowest such winner
can be won in this hand is more than 0.6

As might be expected from a collection of such rules, the performance is not particularly robust, although Carley claims it plays 'quite respectably'. However, it is interesting to note that this approach has survived until very recently, and indeed was the method adopted by the Bridge King program which won the gold medal for Bridge at the 1992 Computer Games Olympiad. There is little description in print about this system, but some comments by Hans Leber — one of its designers — are reported in the 19 September 1992 edition of New Scientist magazine [Geake 92]. He describes the program a little, revealing that it does no look-ahead search at all — a design decision he justifies by claiming that its decisions must be made in 'seconds rather than minutes'. Apparently, 'Bridge King simply deduces as much as it can about where the cards are, then follows a set of rules'. Its level is indicated by the claim that it can 'beat a hobby player, but not a tournament player'. 'Our program still makes silly mistakes,' says Scholz, 'but we feel happy because others make bigger mistakes'.

An interesting attempt to tune the rules in such a system is presented in [Napjus 69]. His approach was to use a machine-learning technique that would 'discover' the best options in any given situation by playing out rival possibilities against a large number of different subdivisions of the outstanding cards. Each possibility (which Napjus referred to as a *line of play* — a concept we will discuss in more detail later) was given a weight and the one which performed better on most of the subdivisions had its weight strengthened slightly. The problem with this approach is that it suffers from a kind of 'Catch-22' situation: in order to test the lines of play open to the declarer, an

algorithm for generating defensive plays is necessary, but defenders judge the selection of their defensive play by reasoning about which lines of play the declarer is likely to be following. In practice, Napjus' system could be easily misled by defence that was very good into thinking that a good line of play was in fact inferior. The learning procedure eventually relied on an algorithm for generating defensive plays that was 'good, but not too good'.

2.2.2 Tactics

Given that the technique of simply using rules has obvious limitations, we now consider how search can be incorporated into card play programs.

The obvious problem when using look-ahead in Bridge is the size of the search space involved. However, one way to avoid examining every possible play sequence in this space is to limit consideration at each stage of planning to a pre-determined set of options. This was the approach taken by Stanier, who constructed plans by linking together *schemas* corresponding to types of plays typically used by human players [Stanier 76]. Stanier's system recognises six schemas: taking top tricks, drawing trumps, ruffing, establishing long cards, finessing, and the promotion of non-masters. However, the exact nature of these schema is never clearly defined and no performance tests or evidence of a working implementation are given.

A more successful attempt at constructing a planner with restricted options is the FINESSE system [Frank 91]. This system restricts itself to considering plays in a single suit, identifying seven types of possible play, or *tactics*. The applicability conditions of each of these tactics are specified as *methods*, which the planner then uses to solve any given problem by generating the entire tree of legal combinations of tactics. Using this technique, a probability of success can be assessed for each option, and later versions of the system also incorporated the ability to generate textual explanations. A typical example of such an explanation [Frank *et al* 92, Page 75] is:

This leads to 2 tricks if West holds at least one of the King or Queen and West holds the ten, or if West holds both the King or Queen, East holds the ten, and East holds the remaining four low cards.

This approach of specifying higher level operators is also employed in the ASPEN sys-

tem [Nygate & Sterling 93], which in fact even uses the same terminology of ‘tactics’. However, ASPEN is less able to distinguish between actual probabilities than FINESSE, and simply attaches one of the three designations *certain*, *usual*, or *possible* to the outcome of any sequence of actions. The general framework of tactics and methods also fits the TIGNUM system developed by [Smith & Nau 93], who explain that ‘to represent the tactical and strategic schemes of card-playing in Bridge we use instances of multi-agent methods’. However, they characterise their system as an example of forward-pruning — a search algorithm which picks *one* choice at any point in the search [Smith & Nau 94].

An extreme application of the use of tactics can be found in the work of [Khemani 94]. For Khemani, ‘the main idea is to retrieve complete plans from memory, which contains the cumulative experience internalised into easily accessible structures’ [Khemani 94, Page 287]. These high-level objects, which Khemani calls ‘Thematic Actions’, are clearly the same notion as FINESSE’s methods. However Khemani’s TA’s typically represent longer strings of actions: play sequences composed of four or five tricks, rather than the single tricks of FINESSE.

In fact, Khemani’s system has no method of comparing the worth of competing options, so it becomes highly important to ensure that ‘best’ matching TA is always returned first for any given problem. However, since this approach necessarily entails constructing a large database of possible situations, it suffers from the obvious drawbacks of ensuring completeness (*i.e.*, that a reasonable play sequence will be selected for every card combination) and of maintaining the integrity of the database to ensure the ‘best’ play for any situation is always the first returned. Further, even if such an approach could eventually be made to play the game of Bridge perfectly it would represent a rather unsatisfactory solution in terms of explaining its actions. Consider, for example, the distinction made by Victor Allis [Allis *et al* 91] between the *cracking* of a game (achieved by producing a computer program which can play perfectly) and the *solving* of a game (which further requires that the algorithm embodied by the program be explicable in human terms). Khemani himself admits that ‘the knowledge contained in the TA cannot provide explanations as to why this play is best’.

The lack of a method for evaluating the worth of plans seems quite a severe limitation,

since there will often be more than one way to make a contract, and in such situations some measure of the plan which is most likely to succeed could prove very important. Further, the object of Bridge declarer play is typically not simply to make a contract, but also to make some overtricks (especially in situations like match-pointed pairs, where the same cards are played by many players and the results compared). Khemani's control system of accepting the first plan which has a chance of making the contract therefore seems overly simplistic. He attempts to justify his approach by arguing that:

Since the cards of all the players cannot be seen, one cannot project moves into the future. Methods like minimax search are therefore ruled out immediately.

However, as we shall see in Chapter 6, this claim represents a substantial over-simplification of the problem.

On close examination, the plans produced by Khemani's system tend to demonstrate some of the weaknesses of his system. For example, in the deal of Figure 2.2, against an opening lead of the J♣ by West, Khemani's system counts the top tricks (there are 11) and then tries to identify suits in which extra tricks could be made.

Contract: South – 7 No Trumps

Lead: West – J♣

	♠ A K 3 2	
	♥ A Q T 9 8	
	♦ J 3 2	
	♣ Q	
♠ J 9 7 5		♠ 4
♥ K 3		♥ J 6 5 4 2
♦ 7		♦ T 9 6 5
♣ J T 7 6 5 2		♣ 9 4 3
	♠ Q T 8 6	
	♥ 7	
	♦ A K Q 8 4	
	♣ A K 8	

Figure 2.2: An example Bridge deal from [Khemani 94, Page 292]

Diamonds obviously provide an extra trick unless the cards are split 5–0 between the defenders, and the spades produce an extra trick if they are split 3–2. Khemani's

system realises this and plans to play for an extra trick in both suits, starting with the spades. After cashing the $A\spadesuit$ and $K\spadesuit$, however, it discovers the spade break, and then re-plans, eventually deciding to re-enter the South hand by winning with the $Q\spadesuit$, and then finessing the $Q\heartsuit$ (the concept of finessing will be described in greater detail in later chapters). Although this results in making the contract, the more natural way of representing this approach would be to explicitly state that the plan being considered is to make extra tricks in the spades and diamond suits, with the heart finesse as a backup source. However, since Khemani's approach is to take the first plan which stands a chance of making the contract, this overall scheme cannot be explicitly represented.

Readers with some Bridge knowledge may also notice that in fact there are ways for declarer to further increase his chances in Khemani's problem. For example, if just *one* of the diamond or spade suits fails to break favourably an extra trick can be produced without resorting to the finesse if the declarer sets up a *simple squeeze*. For example, consider the situation above where the diamonds break but the spades do not. By playing all the winners in spades and diamonds as well as the $K\clubsuit$ and $Q\clubsuit$ declarer can bring the play to the situation of Figure 2.3.

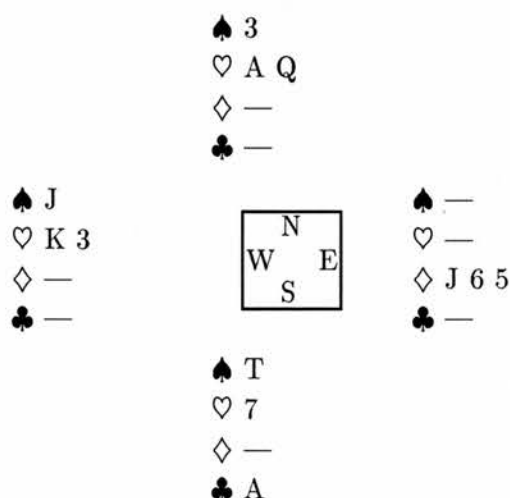


Figure 2.3: Setting up a squeeze in Khemani's example

When South now cashes the $A\clubsuit$, West cannot help allowing the declarer to win the final two tricks. If he plays the $J\spadesuit$, declarer will win with the $10\spadesuit$ and then the $A\heartsuit$. If he plays the $3\heartsuit$, declarer will win with the $A\heartsuit$ and $Q\heartsuit$. This play is called

a squeeze because West is literally squeezed into throwing away a card that he needs to keep. This technique will be investigated in more detail in Chapter 9, but for now the important point to realise is that the play still succeeds even if East and West's cards are swapped in Figure 2.3. We can therefore win whenever the player with the long spades also holds the $K\heartsuit$. A similar squeeze situation can also be set up if it is the diamonds that break unfavourably instead of the spades, and also even if *both* the diamonds and spades break unfavourably, if we can finesse the $Q\heartsuit$.

Such squeeze plays are very subtle, and currently beyond the means of any card play program (although see the discussion of PYTHON in §2.3). This example therefore illustrates the points that Bridge situations are in general difficult to solve, and also that it is not uncommon to find errors in deals presented in Bridge papers. For instance, Figure 2.4 shows the example deal from [Smith & Nau 93] which they claim that their TIGNUM system ‘correctly solved’.

Contract: South – 5♠

Lead: West - 6♦

♠ 4 3 2
♥ 5 4 3 2
♦ 5 4 3 2
♣ 3 2

♠ 10 9 8
♥ Q 9 6
♦ 10 8 6
♣ A 9 8 7

♠ 7
♥ J 10 8 7
♦ Q J 9 7
♣ K Q J 10

♠ A K Q J 6 5
♥ A K
♦ A K
♣ 6 5 4

Figure 2.4: An example Bridge deal from [Smith & Nau 93, Page 90]

This hand is obviously rather artificial, as South's hand is composed of plain winners, except for the three clubs. However, an obvious way to get rid of one of these losers is to ruff it with one of North's trumps. On the lead of the 6♦, TIGNUM wins with the A♦ and then correctly attacks the club suit. Two club tricks are lost, with the defence returning to diamonds each time. After winning East's second diamond continuation

with the 5♠, however, the club ruff is immediately taken. This, of course, risks overruffing by a defender. A better play is to draw two rounds of trumps in case one defender originally started with two clubs and one or two spades.

2.2.3 Tackling Card Combinations

A third way to reduce the search space size in Bridge is to consider only the sub-problems of the *card combinations* in individual suits. This was originally suggested in the domain of Bridge by [Berlin 85]. In Berlin's case, the simplification was made in order to demonstrate a modification to the classical planning framework for deciding how to order possibly interacting subgoals. It is based, however, on the common problem-solving strategy of identifying sub-problems which are solved independently.

The FINESSE system described above benefited from the reduced search space in individual suits (coupled with the use of tactics) to the extent that it could generate the entire tree of legal play sequences for any given card combination. Also, [Kibler & Schwamb 92] make use of the reduction in search space offered by this approach to implement an algorithm for deciding the best moves in single-suit problems. However, this system does not employ tactics, and thus searches through a much larger space (typically in the order of 10^4 compared with the 10^2 of FINESSE).

The problem with tackling card combinations independently, of course, is that they typically interact. Thus, even if sub-plans can be generated for each suit, combining them together into a global plan may not be easy. The dependencies which may exist between the suits give rise to such considerations as *entries*, *unblocking*, and *end-plays* as well as the squeeze plays we examined above. However, human players seem to learn techniques for reasoning about such complications with relative speed. [Gambäck *et al* 91] put forward the suggestion that such problems can be tackled by reasoning about properties such as *resources*. These ideas are not developed beyond a very superficial level, however, and are not implemented. One system which has attempted to implement this approach is that of [Nygate & Sterling 93], described above. This system contains a 'strategy synthesis' module, but very little evidence of performance is included.

2.2.4 Double-dummy Programs

Some authors have tried to simplify the task of automating Bridge card play by solving the easier situation where all the players reveal their cards to each other [Berlekamp 63, Wheen 89]. In this circumstance, the uncertainty over incomplete information is completely removed, and thus the search space of possible moves is also reduced. Since this complete knowledge situation is akin to the opponents placing their cards on the table in the same manner as dummy, this scenario is often described as *double-dummy* Bridge.

The relative ease of solving double-dummy problems has prompted some researchers, for example Levy [Levy 89] and Ginsberg [Ginsberg 95], to suggest Bridge-playing architectures that work by examining a statistically significant number of the possible distributions of the outstanding cards that are consistent with a player's knowledge. They speculate that in any given situation the use of search reduction techniques (such as alpha-beta pruning) would enable the minimax value of each possible action to be established in each of these randomly generated sub-problems, and the best overall action to be predicted by suitably combining these values. In Chapter 6 we present a formalisation that shows that, even in very simple incomplete information games, algorithms like those suggested by Levy and Ginsberg yield suboptimal results against 'best defence'. In particular, we identify two serious problems which afflict such algorithms, independently of how many possible distributions are considered.

2.3 Miscellaneous Bridge Research

There is some Bridge research which does not fit the computer bidder/declarer/defender classification. Nygate, for example, has produced an expert system called PYTHON for identifying squeeze situations [Nygate 84, Nygate & Sterling 90]. As we have already seen, squeezes are an advanced strategy which many human players have difficulty understanding, so even automating their recognition is an achievement. Nygate himself, however, recognises that PYTHON has some important limitations. Firstly, squeeze play is an end-game strategy that normally arises when there are 3 to 7 cards remaining. PYTHON is only capable of recognising such squeeze situations when they are

immediately applicable, producing a plan which creates an extra trick whenever the cards are positioned favourably. Although it was later coupled to a primitive Bridge playing program which could execute these plans [Lustig 85], the system is *not* capable of suggesting what order the cards should be played in to *reach* a potential squeeze situation. Secondly, when the possibility for more than one squeeze play exists, PYTHON cannot choose which one is more likely. This is because it ‘has no knowledge of the distribution of the outstanding cards, in contrast to a Bridge player who can infer a great deal about the adversaries cards’ by clues drawn from the bidding and the play [Nygate 84, Page 45].

Another unusual system is that of [Quinlan 79] — a knowledge-based system capable of making conclusions about card location based on information from the bidding and the opening leads. This system can either locate honours with one of the defenders with certainty, or it can make less specific assertions of the form ‘ x holds either y or z ’. It is not capable of making probabilistic statements, but even so the inferences are of a type that would be useful to a card playing program. Actually producing a system capable of explicitly utilising such inferences would undoubtedly represent a step forward in the state of the art of Bridge programs.

2.4 Commercial Bridge Programs

[Throop 83] traces the development of Bridge-playing programs (mostly those resulting in a consumer product, but also some of a research nature) as far back as 1957. Many of the systems Throop describes achieve a reasonable level of performance, especially in bidding. Unfortunately, however, the descriptions of the algorithms for card play are far from enlightening. For instance, we have already seen Throop’s complete description of one of his own programs at the start of this chapter. Throop makes no attempt to detail the ‘heuristics’ he talks of, leaving readers to imagine for themselves the kinds of algorithms involved. The overall impression is therefore of programs that rely on some kind of pattern-matching to identify common game states, and *ad hoc* modifications to cope with unusual situations.

Since Throop’s book was published, many new commercial programs have been de-

veloped. In order to review these systems, we have gathered together the reviews which have appeared in the specialist computing and Bridge press, and also experimented with some programs ourselves. However, rather than present this review here, we have placed it in Appendix A. The reason for this is partly that the number of programs we examine is very large (over 20), but also because the general summary of our findings simply reinforces the conclusions we have drawn already that whilst some programs may be reasonably good bidders, there are as yet no programs which can play the cards with any degree of expertise. This point is emphasised by, for example, the recent review of computer software in the magazine of the American Contract Bridge League [Manley 94], which found that

While most of the programs received good to excellent ratings on ease of use, graphics, speed and features, scores dropped dramatically when testers rated the skill of the programs - that is, expertise in dummy play and defence. Only three of the nine programs rated above average in skill.

2.5 Comparison with Other Games

The history of serious computer game-playing starts with Shannon's incorporation of the game theoretic results of [von Neumann & Morgenstern 44] into his proposal for a chess playing program [Shannon 50]. The algorithm put forward by Shannon subsequently became known as the minimax algorithm, and was soon adopted by others, for example Samuels, who used it to produce a checkers (8×8 draughts) program [Samuel 59]. Samuels added a learning component to his program and it was quickly able to learn to play better than Samuel himself. Much of the subsequent history of game-playing programs has been devoted to devising other such enhancements to the basic search architecture, with improvements such as alpha-beta pruning, killer heuristics, opening books and end-game databases. More recently the techniques of connectionist computing have also been successfully applied to game-playing situations (for example, see the discussion of TD-GAMMON below).

Some non-trivial games have now been completely solved with the aid of computers. Qubic, for example, was solved by [Allis 92] and Connect-Four by [Uiterwijk *et al* 89] and by [Allen 89]. Interestingly, these two solutions to Connect-Four used complementary approaches: Uiterwijk *et al* employed a knowledge-based approach to deduce the

value of any position whereas Allen used brute-force depth-first search. The former, knowledge-based approach led to the formalisation of nine rules which, if followed, would result in the game being played perfectly. The contrast between this and Allen's brute-force approach in part formed Allis' motivation for distinguishing between the cracking and the solving of a game discussed previously.

Other games, whilst not being cracked or solved by computers, have seen domination slip away from the human players. In Othello, for example, the Santa Cruz Machine Othello Tournament in 1982 prompted the then human World Champion Jonathan Cerf (quoted in [Rosenbloom 82]) to write that:

In my opinion the top programs from Santa Cruz are now equal (if not superior) to the best human players.

The winner of this contest was IAGO [Rosenbloom 82]. However, a more recent program called BILL [Lee & Mahajan 88] incorporates a scheme for learning evaluation functions and is even more impressive. On a collection of endgame problems in which human experts had an accuracy of 47.4%, BILL's success rate was 68%. In a contest against Brian Rose, the then highest-rated U.S. Othello player, it won resoundingly by a margin of 56 games to 8. It seems undeniable that computers now perform at World Championship level in Othello.

Another game in which computers are on a par with the best humans is Backgammon. In 1980, Berliner's BKG9.8 program beat the then World Champion in an exhibition match by a score of 7 games to 1 [Berliner 80]. Although this victory was partly due to luck, more consistent computer programs have recently emerged. In particular, the use of neural networks by [Tesauro & Sejnowski 89] has led to the development of TD-GAMMON, which has been estimated to play at -0.05 points per game against the noted and highly respected former World Champion Bill Robertie, making the program the best anywhere, and one of the strongest players in the world [Tesauro 94].

The first program to actually play for a World Championship was the CHINOOK checkers playing program [Schaeffer *et al* 92]. This was a right which the program won by placing second behind the World Champion Marion Tinsley in the 1990 Checkers U.S. National Open. In his title defence, Tinsley won by a score of 4 wins to 2, with 33 draws [Schaeffer *et al* 93b]. However, this still represents a very creditable performance

by CHINOOK, since Tinsley had the unparalleled record prior to the meeting of only losing 7 games in the past 42 years.¹

In chess, computers have not yet managed to challenge for the world title, although the well-known DEEP THOUGHT program has been given a rating of 2552 by the U.S. Chess Federation, placing it in the bottom half of the grandmaster range [Hsu *et al* 90]. However, its designers are optimistic [Hsu *et al* 90, Page 50] that the trend of increased speed leading to an increased rating will enable their next generation system, with a 1000-fold increase in speed, to

play at a 3400 level, about 800 points above today's DEEP THOUGHT and 500 points above Kasparov's rating record.

This speculation is questioned, though, by the team designing CHINOOK. CHINOOK searches to a *minimum* depth of 17-, 19-, and 21-ply (one ply is one move by one player) in the opening, middlegame, and endgame respectively, and uses a selective deepening heuristic that results in major lines of play being searched to 30-ply or more. In checkers, however, it appears that the advantage gained by searching an extra ply at such depths is very small. For example, during CHINOOK's title play-off with Tinsley each player made about 1000 moves, and since all the games which led to a result could be traced to a single mistake by one of the players, it can be estimated that their accuracy when choosing a move was 99.6% and 99.8% respectively. In fact, the errors made by CHINOOK against Tinsley would not have been prevented even with an extra 10-ply of search! (See Figure 2.5.)

An example of these deep search problems comes from an exhibition match against Tinsley in 1990. CHINOOK realised it was in difficulty on move 25 and the programmers resigned on its behalf on move 35. After the game, however, Tinsley revealed that 'on move 12 he had seen to the end of the game and knew he was going to win'. The best estimate of the CHINOOK team was that this feat involved 'selective searches of over 60 ply'. Even in one of CHINOOK's victories from the 1992 World Championship, Tinsley said he realised he was lost immediately after making the fatal move. CHINOOK, on the other hand, took an additional 12 moves (24 ply) before a 21-ply search found the

¹ Sadly, Marion Tinsley has since died, on April 17 1995.

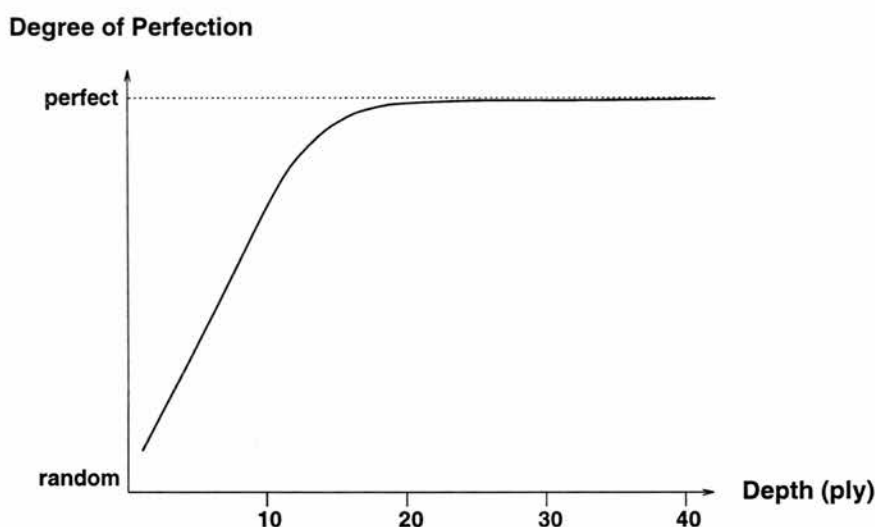


Figure 2.5: CHINOOK’s performance as a function of depth (after [Schaeffer *et al* 93a, Page 54])

win. That is, ‘although CHINOOK found the best moves...it did so without knowing it was winning’ [Schaeffer *et al* 93a, Page 55].

It is unclear exactly how these results carry over to the game of chess, although [Schaeffer *et al* 93a] speculates that both games follow similarly-shaped curves to that of Figure 2.5. With the current top chess programs searching about 10-ply, it could be that the performance of the next generation of chess computers will not be as strong as their developers hope. The implications of this for Go, where ‘there is no program in the world which can defeat an amateur of moderate strength in an even game’ [Erbach 92] have also yet to be seen.

2.6 Summary

We have looked at literature on the game of Bridge, examining both academic research and commercial products. The important conclusion for this thesis is that although expert level performance has been achieved in the sub-problem of Bridge bidding, the overall standard of computer Bridge play in general is very low. Especially when compared to the success of computers in other games, this is surprising. In games

like chess and checkers, computer programs are amongst the top players in the world. In Bridge, however, a history of academic research and a proliferation of more than two dozen commercial software packages has failed to produce a system capable of competing with even good novice human players. Despite the protestations by Death² at the beginning of the chapter that 'THEY ONLY LIVE EIGHTY OR NINETY YEARS!', then, humans appear to make very good Bridge players. In Chapter 4 and Chapter 6 we will identify some of the reasons why this might be so.

² Death always speaks in capital letters.

Chapter 3

Planning Literature

What are hopes, what are plans?

— JOHANN CRISTOPH FRIEDREICH VON SCHILLER
Die Jungfrau von Orleans, act I, sc. 2

What are plans? In this chapter, we answer this question by looking at AI planning systems. Such systems use *operators* to represent the possible *actions* that can be carried out in a domain. The ultimate aim of planning is then typically to find a *ground operator sequence* (*i.e.*, a sequence with no variables) which, when executed in the initial state of some problem, will produce the desired *behaviours* or sequences of states in the domain. This is relevant to Bridge since declarer play involves playing cards in such a way as to make a specific number of tricks.

We also introduce refinement search — a formalism recently employed in [Kambhampati 94, Kambhampati *et al* 95] as a unifying framework within which to cast and compare various planning algorithms. The nature of this framework facilitates the comparative analysis of existing planning systems, and will also allow us to describe the nature of the FINESSE system more clearly in later chapters. To make use of this formalisation, we set out the basics of refinement search and then examine how some of the historical planning systems we discuss may be viewed as different flavours of the basic algorithm.

3.1 Plan-space and State-space

There are a large number of planning systems in the AI literature. To help us to make sense of the diversity we therefore begin by establishing some terminology.

3.1.1 Plan Representation

It is possible to distinguish two types of plan representation schemes used by AI planning systems. Firstly, there is *state-space* representation, in which a plan is composed of *states* — representations of the problem at each stage of its solution — and *operators* which transform one state to another. State-space plans are easily represented on a computer, for example using directed graphs with nodes describing the states and arcs describing the application of the operators that make the appropriate transformations between the states. With this kind of encoding, a solution is just a path which starts at the goal and finishes at one of the leaf nodes that corresponds to a solution of the problem.

A different representation is the *action-ordering* representation. This kind of plan usually has nodes which correspond to actions and branches that determine ordering relations between the actions. The advantage of this type of representation is that two operators can be ordered within a plan without the need to generate, or even specify, the intermediate states arising between the execution of each. However, the drawback, compared to a state-space representation, is that it becomes more difficult to determine when a plan is actually a solution to a problem. As the state of the domain is no longer explicitly present in the plan representation, the achievement of the goal behaviours must be checked by reasoning about the possible orderings of the plan's operators.

In either of these types of plan representation, the ordering over the operators need not be fully specified. That is, it can be possible for two or more operators to be executed in any order or even at the same time. Plans which allow this are called *partially ordered* and planners constructing such plans are called *partial order planners* (or sometimes *non-linear planners*, since the emerging plan resembles a network rather than a linear sequence).

3.1.2 Search Techniques

The possible search regimes of planners also come in two main flavours that broadly mirror the distinction between the types of plan representation. Firstly, it is possible to take the initial state of the problem as a start state and apply operators until a goal

state is reached. In such a planner, the nodes of the search space correspond to states and the arcs to possible operators. This type of planner therefore searches through all the possible states, and planners adopting this approach are described as *state-space planners*. The benefits of this approach are described by [Drummond & Tate 87, Pages 9–10]

...the search procedure is almost all the *reasoning* that's required of the planning system. The reasoning required to build a plan is packaged into the planner's next-stage generator. This means that the planner will only generate *possible* plans, and need perform no check after extending a plan.

The main alternative possibility to state-space planning is *plan-space planning* in which each node of the search space contains *partial plans*. Initially, there are typically two 'null' operators in the plan which correspond to its *start* and its *finish*. The *start* operator is used to assert the starting conditions of the plan by specifying them as the operator's postconditions. The *finish* operator asserts the goal conditions by specifying them as the operator's preconditions. At each stage of planning, operators are added or orderings imposed on hitherto unordered operators until a plan is produced that is guaranteed to work. [Drummond & Tate 87, Page 10] summarises one distinction between state-space and plan-space planning.

When the planner searches through [the space of] states, plan modification is only possible at the 'tail end' of the plan. Operators are added to the plan by trying out another operator application, and operators are removed from the plan when backtracking occurs. This is a result of confusing the plan construction reasoning with search space navigation control. However, if the planner searches through a space of partial plans, it can add and remove operators wherever in the plan it sees fit.

The drawback to the plan-space planning approach is that the task of determining when a plan actually represents a solution becomes much harder. It is no longer the simple matter of checking the current state against the goal state, but instead calls for reasoning about whether the operators and constraints of the plan are sufficient to guarantee the goal state being produced when the plan is executed. We will examine this issue in more detail below.

3.2 Planning Systems

Before introducing refinement search, we look briefly at some of the AI planning systems to date. For a more detailed survey, readers are referred to literature such as [Drummond & Tate 87].

The most influential single planning system is probably STRIPS, produced in 1972 [Fikes *et al* 72]. This is a state-space planner that applies the principle of *means-ends analysis* (MEA), first introduced by the General Problem Solver (GPS) project [Ernst & Newell 69]. MEA is a search heuristic which works by selecting operators that can be shown to be significant in reducing the differences between the current state in the search space and the desired state. However, differences are reduced one by one, which means that the heuristic will not always be able to correctly solve some problems. The reason for this is described by [Simon 83, Page 15]:

The search procedure of GPS is built on the implicit premise that if the present situation differs from the goal situation by features A, B, C, \dots , then the goal situation can be attained by removing the differences A, B, C, \dots , in some order. Of course this premise is false unless the matrix of connections between differences and operators can be triangularized. This matrix can be triangularised just under those conditions when an appropriate composition axiom would be valid in the modal logic; that is, just when there is independence among the actions.

This assumption of independence means, for example, that MEA will not be able to solve the *Sussman anomaly* in which solving one subgoal undoes the achievements of the plan for solving the other. This anomaly is from the domain of blocks worlds, and involves trying to make a stack of three blocks with the goals $(On\ A\ B) \wedge (On\ B\ C)$. Forming and executing separate plans for each of the subgoals in this problem, however, does not guarantee achieving them both because the initial conditions of the plan are such that, whichever subgoal is established first, the operators for establishing the second subgoal will undo the first subgoal in the process. Sussman's own planner, HACKER, was also unable to solve this problem, since it assumed [Sussman 73, Page 58] that 'Subgoals are independent and thus can be sequentially achieved in an arbitrary order'. This assumption of independence explicitly removes the solution from the search space.

In fact, to produce a solution to the Sussman anomaly, the operators for achieving each subgoal have to be interleaved together: assuming that the subgoals can be achieved independently prevents this from happening. Other plan-space planners such as INTERPLAN [Tate 75] and WARPLAN [Warren 76] *could* solve the Sussman anomaly because they did not assume independence of subgoals. NOAH, the first partial-order planner [Sacerdoti 75], could also solve the problem, but mostly because of its choice of ‘goal to work next on’ and not its use of partial ordering.

NOAH performed little (if any) search, as it had no means of backtracking from decisions that did not lead to a solution. This was partly because of the relative difficulty in partial-order plan-space planning, noted above, of deciding whether the preconditions of any given operator hold at a particular point. The first system to demonstrate this ability to any degree was NONLIN [Tate 77], which used a procedure called *question answering* (QA) to determine the truth of operator preconditions. This is described by Drummond [Drummond & Tate 87, Page 11]:

If the precondition in question is true, the QA simply returns *true*. If the precondition is false, then QA returns a description of how the precondition can be made true. The alternatives include the addition of operators, the addition of orderings and the bindings of variables within the developing plan.

Chapman formalised some of what NONLIN does, producing the TWEAK planner based on his observations [Chapman 87]. TWEAK checks the validity of its plans with a *modal truth criterion* (MTC) that specifies the necessary and sufficient conditions for a condition to be true at a point in a partially ordered plan. However, Chapman also shows that the truth criterion of any non-linear planner will be NP-complete if the operators it uses are powerful enough to represent actions whose outcome depends on their input situation. Despite this, some planners have evolved which allow such representations. SIPE, for example has operators which may have *context-dependent* effects [Wilkins 84, Wilkins 88]. In order to use such operators efficiently, SIPE incorporates heuristics. For example, the truth of the ‘main effects’ of an operator are always checked explicitly, but [Wilkins 89, Page 5]:

for other query predicates the system merely proves that there is one possible ordering of the partially ordered actions that makes the query predicate true, without enforcing that order. This is efficient, but can produce

temporarily invalid plans, since different calls to the truth criterion may assume different implicit orderings.

Another technique for coping with context-dependent effects are the *secondary preconditions* of [Pednault 88, Pednault 91]. These preconditions are used to define the conditions under which actions actually have their desired effects. An example of a planner using this approach is UCPOP [Penberthy & Weld 92], which is an extension of the planner SNLP [Barrett *et al* 91, McAllester & Rosenblitt 91].

SIPE, as well as NOAH, NONLIN and other more recent planners such as O-PLAN [Currie & Tate 85] are examples of *Hierarchical Task Network* (HTN) planners. In such planners, the nodes do not have to correspond to simple domain actions and can instead represent higher-level ‘tasks’, such as ‘Paint the ceiling’ or ‘Build a house’. One of the planning techniques employed by such planners is then the notion of ‘task reduction’ in which the higher-level tasks are broken down to a lower level. HTN planning continues until all the nodes in the plan correspond to simple domain actions.

The problem of planning under uncertainty has also been tackled, notably by a family of planners related to SNLP. For example, SENSP [Etzioni *et al* 92] and CASSANDRA [Collins & Pryor 95, Pryor 95] build on UCPOP (which we noted above is based on SNLP), while CNLP [Peot & Smith 92] is based on SNLP directly. The first two of these use the context-dependency of operators to represent actions with uncertain outcomes, while SENSP uses a three-valued logic, and allows actions to have multiple, mutually exclusive sets of outcomes. The different possible outcomes of the uncertain actions contained in these systems’ plans can be referred to as *contingencies*, and in general, ‘the aim of contingency planning is to construct a single plan that will succeed in all circumstances’ [Pryor & Collins 96]. BURIDAN [Kushmerick *et al* 95], a further planner based on SNLP, is probabilistic, and tries to construct plans that have a high probability of success. BURIDAN itself cannot construct contingency plans (*i.e.*, plans containing alternatives courses of action for different circumstances), but it has been extended into a probabilistic contingency planner in the C-BURIDAN system [Draper *et al* 94b, Draper *et al* 94a], which also draws on CNLP. Another contingency planner strongly influenced by CNLP is PLINTH [Goldman & Boddy 94a, Goldman & Boddy 94b] which is a total order planner based on McDermott’s PEDESTAL [McDermott 91].

Other approaches to planning under uncertainty include interleaving planning with execution (for example, IPEM [Ambros-Ingerson & Steel 88] and SAGE [Knoblock 95]), and reactive planning, in which a set of condition-action rules is generated instead of a branching plan containing specific actions (the *universal plans* of Schoppers [Schoppers 87], or Situated Control Rules [Drummond 89]).

3.3 Refinement Search

As we will see in Chapter 8, one of the techniques used by FINESSE to produce plans for the game of Bridge is partial-order plan-space refinement. We will therefore look at this type of planning more closely. To do this, we will present a formalisation of such systems introduced by [Kambhampati 94] and more recently updated in [Kambhampati *et al* 95]. This formalisation uses refinement search to describe partial-order plan-space planners. More recently, Kambhampati has extended this to further incorporate state-space planners and HTN planners. Readers interested in these developments are referred to [Kambhampati & Srivastava 95] and [Kambhampati 95], respectively.

3.3.1 Background

Refinement search is essentially a particular view of the *generate-and-test* paradigm first discussed by Newell and Simon in their Turing Award Lecture [Newell & Simon 76]. Rather than employing a search process that involves the gradual *construction* of a solution, refinement search starts with the *complete set* of possible sequences of actions, and then gradually subdivides it into smaller subsets by a process of repeated splitting (or branching, or *refinement*). Thus, at each stage of the search, the overall set of possible solutions will be reduced to a smaller subset — the *solution candidates*. Of course, a solution (if one existed) could have been found initially by simply conducting an exhaustive check on each member of original complete set of possibilities, but the size of this set usually makes this impractical. Refinement search generally continues until an actual solution can be picked out from the solution candidates in *bounded* time.

Since the number of solution candidates at any search node may be enormous, or infinite, in practice they are never represented explicitly. Instead each set is typically implicitly described by a set of constraints, so that any sequence of actions that is consistent with the constraints is taken to be a solution candidate of that node. Under this interpretation, refinement search involves starting with a null constraint set, and then systematically adding new constraints at each step until a solution can be identified.

3.3.2 Basic Formalisation

A slightly modified form of the refinement search algorithm introduced by Kambhampati is reproduced in Figure 3.1. It is specified by providing the following:

- G , the solution criterion of the problem (the conditions a solution must meet),
- R , a set of refinement operators (or strategies), and
- `sol` the *solution constructor function*; a 2-place function that takes as its argument a search node N and the solution criterion G , returning one of three values:
 - `*fail*`, meaning that none of N 's solution candidates is a solution to the problem.
 - Some solution candidate of N which does satisfy the solution criterion G .
 - \perp , meaning that `sol` can neither find a candidate that is a solution, nor determine that no such candidate exists.

Search is initiated by calling *Refine-Node* on a node with a null set of constraints and then progresses, through the application of refinement operators, until a node is found for which the solution constructor function returns a solution.

3.3.3 Brief Discussion

As an example to illustrate refinement search, consider the well-known '8 Queens Problem' of placing 8 queens on a chess board in such a way that none of them attack any of the others. This problem can be approached by starting with an empty board and then

Algorithm *Refine-Node*(\mathcal{N})

Parameters: *sol*, the solution constructor function
 \mathbf{R} , Refinement operators
 G , Solution criterion

- 0 **Termination check.** If *sol*(\mathcal{N}, G) returns a candidate c , then return it and terminate. If it returns **fail** then fail. Otherwise continue.
- 1 **Refinement.** Pick a refinement operator $r \in \mathbf{R}$. (Not a backtrack point.)
 Nondeterministically choose a refinement \mathcal{N}' from $r(\mathcal{N})$, the refinements of \mathcal{N} with respect to r .
- 2 **Consistency check** (Optional). If \mathcal{N}' is inconsistent (*i.e.*, the node has *no* solution candidates; its constraint set is unsatisfiable), fail. Else continue.
- 3 **Recursive invocation.** Recursively invoke *Refine-Node* on \mathcal{N}' .

Figure 3.1: A generic refinement search algorithm (after [Kambhampati *et al* 95, Page 172])

repeatedly placing a queen in a random but legal position. Viewed from a generate-and-test perspective each partially filled board in this search process would represent a building block in the construction of a completed solution that would include the pieces already in place. From a refinement search perspective, however, the partially filled board is actually a representation for the (perhaps very large) set of all the possible boards that contain its configuration as a sub-grouping. So, in refinement search, the partial solution itself contains (by virtue of representing a set of solutions) the eventual solution, whereas in a generate-and-test approach the roles are reversed, with the eventual solution containing each partial solution generated during its construction.

The refinement search framework became popular in the field of operations research, and it is this difference in pedigree with the generate-and-test paradigm that accounts for their differing outlooks. In operations research, great importance was attached to theoretical guarantees of optimality and completeness. AI, on the other hand, has ‘always emphasised the heuristic and implementational aspects of problem solving and its parity with human style over its mathematical foundations’ [Pearl 84]. In AI, acts such as adding a Queen to a board in the 8-Queens problem, or the extension of a path in a search for a solution to the 8-puzzle, are naturally thought of as *creative* steps in

the search for a solution, rather than as deletions in a process of refinement. [Pearl 84], however, points out that

The split-and-prune paradigm emphasises the commonality of problem-solving methods over many problems, whereas the object creation paradigm focusses on the unique structure featured by a specific symbolic representation of a given problem.

This focus on the common aspects of problem-solving algorithms can be exploited, as we shall see in the following sections, to emphasise both the points of similarity and the areas of divergence of different algorithms, as well as enabling the clear presentation of the place and function of new enhancements.

3.4 Plan-space Planning as Refinement Search

We now examine how plan-space planning can be cast as refinement search, drawing on [Kambhampati 94] and [Kambhampati *et al* 95].

3.4.1 Partial Plan Representation

The first task when describing plan-space planning in a refinement search framework is to decide how to represent a partial plan. Kambhampati uses a 5-tuple of quantities for this purpose which we describe below.

Definition 3.1 (Partial Plan) *A partial plan is a 5-tuple $\mathcal{P}: \langle T, \mathcal{O}, \mathcal{B}, ST, \mathcal{L} \rangle$ where:*

- T is the set of steps in the plan.
- \mathcal{O} is a partial ordering relation over T .
- \mathcal{B} is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and postconditions of the operators.
- ST is a symbol table which maps steps to ground operators in the domain.
- \mathcal{L} is a set of auxiliary constraints. (See §3.4.2.)

Before introducing auxiliary constraints, we will first examine a simple example plan without this complication. Figure 3.2 gives an example of a plan which contains the four steps t_0 , t_1 , t_2 , and t_∞ . There are three operators in the domain with preconditions and postconditions (the facts ‘add’ed or ‘del’eted by the operators) as shown in the figure. The four steps in the plan are mapped to the operators **start**, o_1 , o_2 , and **fin** respectively, and the ordering constraints are as shown in the standard network representation.

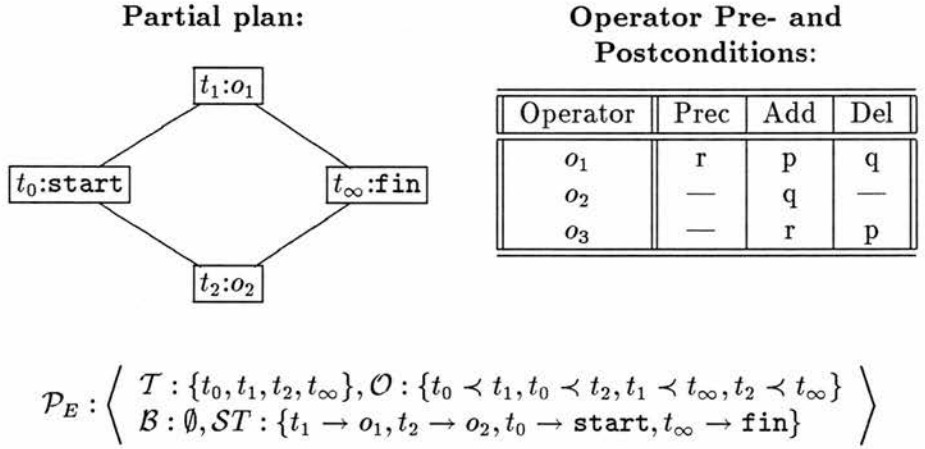


Figure 3.2: A simplified partial plan (\mathcal{P}_E) in a domain with three operators (after [Kambhampati *et al* 95, Pages 177-178])

Recall from the start of this chapter that we described the main aim of planning as being the generation of a ground operator sequence which, when executed in the initial state of some problem, will produce the desired behaviours or sequences of states in the domain. To see how the representation of Figure 3.2 represents a set of possible ground operator sequences we can define the following mapping function [Kambhampati *et al* 95, Page 179]:

Definition 3.2 (Mapping Function) \mathcal{M} is said to be a mapping function from a plan \mathcal{P} to a ground operator sequence \mathcal{S} if

- \mathcal{M} maps all steps of \mathcal{P} (except the dummy steps t_0 and t_∞) to elements of \mathcal{S} , such that no two steps are mapped to the same element of \mathcal{S} ,
- \mathcal{M} agrees with ST (i.e., $\mathcal{M}(t) = ST(t)$ for all steps, t , of \mathcal{P}), and

- for any two steps t_i, t_j in P such that $t_i \prec t_j$, if $\mathcal{M}(t_i) = \mathcal{S}[l]$ (i.e., \mathcal{M} maps t_i onto the l th operator in \mathcal{S}) and $\mathcal{M}(t_j) = \mathcal{S}[m]$ then $l < m$.

Thus, for our example plan of Figure 3.2 *any* ground operator sequence which contains the operators o_1 and o_2 will be a solution candidate of the plan, since there are no ordering relations between t_1 and t_2 in \mathcal{P}_E , and we simply need to specify \mathcal{M} so that it maps t_1 and t_2 onto (one of) the occurrences of the operators o_1 and o_2 , respectively. (Notice that the distinction between the steps in the plan and the actual domain operators enables sequences in which some operators are repeated to be easily accommodated.)

An important concept which can also be defined on just the simplified representation of Figure 3.2 is that of a *ground linearisation* [Kambhampati *et al* 95, Page 178]:

Definition 3.3 (Ground Linearisations) *A ground linearisation of a partial plan $\mathcal{P}: \langle T, \mathcal{O}, B, ST, \mathcal{L} \rangle$ is a fully instantiated total ordering of the steps of \mathcal{P} that is consistent with \mathcal{O} (i.e., a topological sort) and B .*

A ground linearisation captures ‘the syntactic notion of what it means for a partial plan to be consistent with its own ordering and binding constraints’ [Kambhampati *et al* 95, Page 179]. For the example plan of Figure 3.2 there are two ground linearisations: $t_0 t_1 t_2 t_\infty$ and $t_0 t_2 t_1 t_\infty$. Ground linearisations go part of the way towards addressing how to produce ‘desired behaviours’ from sequences of actions. In order to see the full story, we now need to consider the auxiliary constraints.

3.4.2 Auxiliary Constraints

According to [Kambhampati 94, Page 332], ‘almost all of the auxiliary constraints employed in classical planning can be formalised in terms of two primitive types of constraints’. The first of these — *interval preservation constraints*, or IPCs — constrain all the operators that occur between two specific points in a plan to preserve some specified condition. Kambhampati’s formal definition of how some ground operator sequence \mathcal{S} can satisfy an IPC of a partial plan is given below, and in Figure 3.3 on the next page we also supply a pictorial representation to aid visualisation.

Definition 3.4 (Interval Preservation Constraints) An interval preservation constraint $\langle t_i, c, t_j \rangle$ of a plan \mathcal{P} is said to be satisfied by a ground operator sequence S according to a mapping function M that maps steps of \mathcal{P} to elements of S , if and only if every operator o in S that comes between $M(t_i)$ and $M(t_j)$ preserves the condition c (i.e., if c is true in the state before o , then c will be true in the state resulting from its execution).

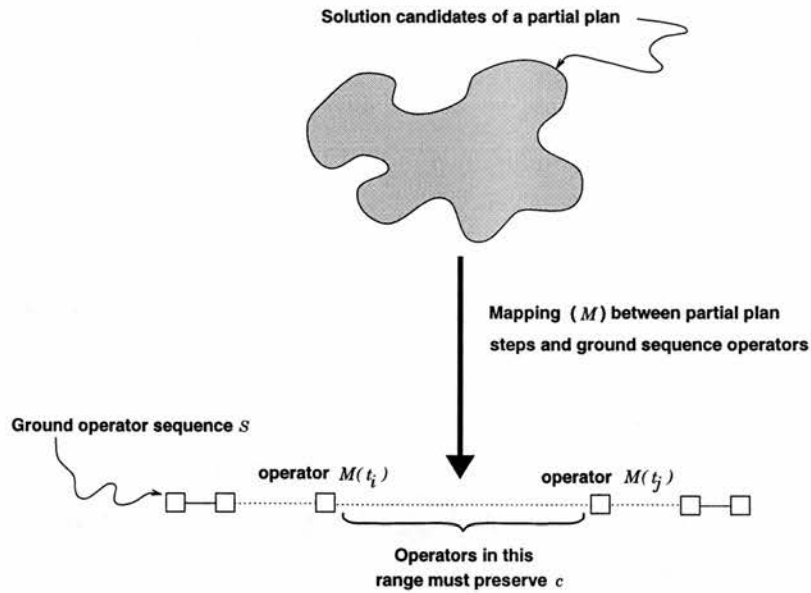


Figure 3.3: How a ground operator sequence S can satisfy an IPC $\langle t_i, c, t_j \rangle$

The second type of constraints are *point truth constraints*, or PTCs, which as their name suggests, concern constraints which must be true at just one particular point in a plan. Kambhampati's definition of PTCs [Kambhampati 94, Page 332] is as follows:

Definition 3.5 (Point Truth Constraints) A point truth constraint $\langle c@t \rangle$ is said to be satisfied by a ground operator sequence S with respect to a mapping M that maps steps of \mathcal{P} to elements of S , if and only if either c is true in the initial state and is preserved by every action of S occurring before $M(t)$, or c is made true by some action $S[j]$ (i.e., the j th element in the operator sequence) that occurs before $M(t)$, and is preserved by all the actions between $S[j]$ and $M(t)$.

To see how these constraints can be used to make a plan achieve a desired goal, consider Figure 3.4 which gives a more complete version of our original simple example of a partial plan. Let us assume that the initial state of the world in which this plan will be executed is null and that the top-level goals are p and q . These goals are represented by including the two PTCs $\langle p@t_\infty \rangle$ and $\langle q@t_\infty \rangle$ as the preconditions of the final step. Now imagine that the steps t_1 and t_2 have been introduced to achieve these goals. In order to ensure that no other operator ‘undoes’ these goals after they are achieved, the two IPCs $\langle t_1, p, t_\infty \rangle$ and $\langle t_2, q, t_\infty \rangle$ can be added to the plan, as shown. Also, because operator o_1 has r as its own precondition, the PTC $\langle r@t_1 \rangle$ is further added. Any ground operator sequence satisfying these constraints will be a solution to the problem.

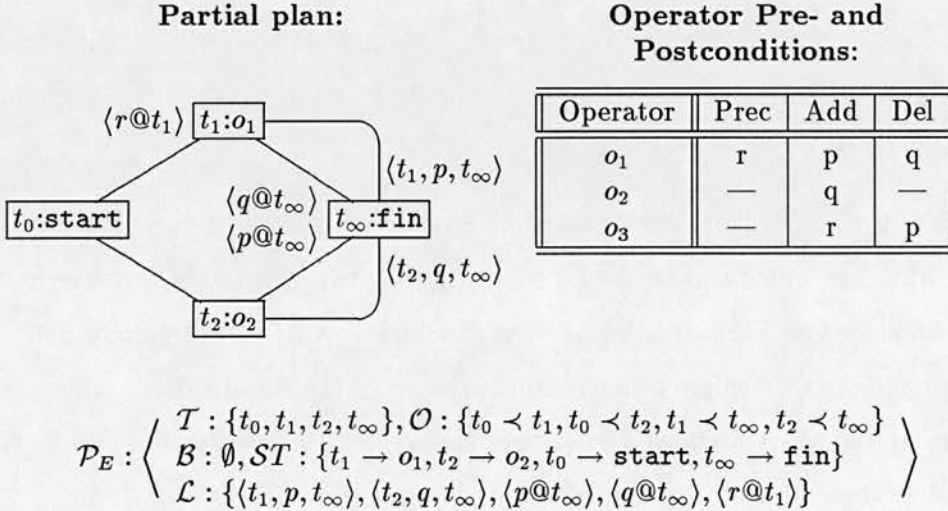


Figure 3.4: The full representation of the partial plan \mathcal{P}_E (after [Kambhampati *et al* 95, Page 177-178])

3.4.3 A General Plan-space Planning Algorithm

Although the actual solutions to a planning problem at some node \mathcal{P} will have to obey all of the auxiliary constraints of \mathcal{P} , during the search process it will be useful to make a distinction between two types of constraint: those that are *monotonic*, and those that are not [Kambhampati 94, Page 332]:

Definition 3.6 (Monotonicity of Auxiliary Constraints) *An auxiliary constraint C is monotonic if, given a ground operator sequence S that does not satisfy C , no operator sequence S' obtained by adding additional ground operators to S will satisfy C .*

Interval preservation constraints are monotonic. To see this, look again at the example of Figure 3.3 on Page 45. If it is the case that S does *not* satisfy the constraint in this diagram, then (at least) one of the operators between $\mathcal{M}(t_i)$ and $\mathcal{M}(t_j)$ must be violating the condition c . The act of adding extra operators to S , no matter what position they are inserted, cannot change the fact that the protected condition is violated in the protected range, so the constraint is monotonic.

Point truth constraints, on the other hand, are non-monotonic. If some PTC $\langle c@t \rangle$ is not satisfied by an operator sequence S , the constraint can still be made true by adding extra operators (for example, by inserting an action that makes c true immediately prior to t).

This distinction is useful because the monotonic constraints provide a useful pruning mechanism which can be used during planning. Specifically, if the search ever reaches a node at which none of the solution candidates satisfy all the monotonic constraints, the node can be safely pruned; no extensions of the linearisations will ever be able to satisfy the constraints, since they are monotonic. To make use of this feature, we will define the following concept [Kambhampati *et al* 95, Pages 179&183]:

Definition 3.7 (Safe Ground Linearisation) *A ground linearisation G of a plan \mathcal{P} is said to be **safe**, if it satisfies all the monotonic auxiliary constraints. In our representation, the monotonic constraints are all IPCs. G satisfies an IPC $\langle t, p, t' \rangle$ if for every step t'' between t and t' in G , the operator $ST(t'')$ does not delete p .*

Let us consider the ground linearisation $G_1 : t_0 t_1 t_2 t_\infty$ in our example of Figure 3.4. G_1 satisfies the IPC $\langle t_1, p, t_\infty \rangle$ since the only step between t_1 and t_∞ is t_2 , and the operator $ST(t_2)$ is o_2 , which does not delete p . G_1 also trivially satisfies the IPC $\langle t_2, q, t_\infty \rangle$, since there are no steps between t_2 and t_∞ . G_1 is therefore a safe linearisation. The linearisation $G_2 : t_0 t_2 t_1 t_\infty$, on the other hand, is not safe since it breaks the constraint

$\langle t_2, q, t_\infty \rangle$; the operator o_1 corresponding to step t_1 deletes q .

The Algorithm

Kambhampati's planning algorithm is reproduced in Figure 3.5 on the following page. For simplicity, this algorithm is designed to process just the PTC's of a partial plan until they are guaranteed to be satisfied. It therefore only really solves problems whose goals are *goals of attainment* (since, as we have seen, the goal of achieving some condition p can be expressed as the PTC $\langle p, t_\infty \rangle$). However, other types of goals giving a richer behaviour could also be modelled. For example, *maintenance goals* could also be handled by imposing interval preservation constraints on the initial plan. Also, *intermediate goals* could be handled by introducing dummy steps along with PTC's stating the goals that should be achieved at these steps.

To keep track of the PTC's yet to be specifically addressed by the planner, an agenda is used. During planning, the preconditions from this agenda are successively examined and removed, after extra orderings or actions have been introduced. Any extra PTC's introduced are generally added to both the agenda and to the list of auxiliary constraints, but notice that if there are conditions that we don't want the planner to work on explicitly (Pednault [Pednault 91] gives the example of not wanting to build an airport for the express purpose of visiting a city as an example of such *filter* conditions) they may be added only to \mathcal{L} .

Given a problem with the set G of goals (of attainment) planning is invoked by calling *Refine-Plan* with the 'null' partial plan and agenda:

$$\mathcal{P}_0 : \left\langle \begin{array}{l} \mathcal{T} : \{t_0, t_\infty\}, \mathcal{O} : \{t_0 \prec t_\infty\}, \mathcal{B} : \emptyset, \\ ST : \{t_0 \rightarrow \text{start}, t_2 \rightarrow \text{fin}\}, \mathcal{L} : \{\langle g_i @ t_\infty \rangle | g_i \in G\} \end{array} \right\rangle$$

$$\mathcal{A}_0 : \{\langle g_i @ t_\infty \rangle | g_i \in G\}$$

Planning continues until the solution constructor function can identify a solution. Kambhampati identifies the solution constructor function *all-sol* which terminates only when *all* the ground linearisations of a partial plan correspond to solutions [Kambhampati *et al* 95, Pages 179&189].

Algorithm *Refine-Plan*($\langle \mathcal{P}: \langle T, \mathcal{O}, B, ST, \mathcal{L} \rangle, \mathcal{A} \rangle$)

Parameters: *sol*, the solution constructor function

G , the set of goals of the problem.

pick-prec, routine for picking preconditions from the plan agenda for establishment.

interacts?, routine used by pre-ordering to check if a pair of steps interact.

conflict-resolve, routine for resolving monotonic auxiliary constraint conflicts.

0 **Termination check.** If *sol*(\mathcal{P}, G) returns a solution, return it, and terminate. If it returns *fail*, fail. Otherwise continue.

1 **Refinement.** Refinements fall into two broad classes:

- **Establishment Refinement.** Refine the plan by selecting a goal, choosing a way of establishing that goal and optionally remembering the establishment decision:

1.1 **Goal Selection.** Using the *pick-prec* function, pick from \mathcal{P} a goal $\langle c, s \rangle$ to work on. *Not a backtrack point.*

1.2 **Goal Establishment.** Non-deterministically select a new or existing establisher step s' for $\langle c, s \rangle$. Introduce enough ordering and binding constraints, and secondary preconditions to the plan such that (i) s' precedes s (ii) s' will have an effect c , and (iii) c will persist until s (i.e., c is preserved by all the steps intervening between s' and s). *Backtrack point; all establishment possibilities need to be considered.*

1.3 **Book Keeping (Optional).** Add auxiliary constraints noting the establishment decisions, to ensure that these decisions are protected by any later refinements. This in turn reduces the redundancy in the search space. The protection strategies may be one of *goal protection*, *interval protection*, and *contributor protection*. The auxiliary constraints may be either point truth constraints or interval preservation constraints.

- **Tractability Refinements (Optional).** Help in making the plan-handling and consistency check tractable. Use either one or both:

1.4 **Pre-ordering:** Impose additional orderings between every pair of steps of the partial plan that possibly interact. according to the static interaction metric *interacts?*. *Backtrack point; all interaction orderings need to be considered.*

1.5 **Conflict Resolution:** Add orderings, bindings and/or secondary (preservation) preconditions to resolve conflicts between the steps of the plan, and the plan's monotonic auxiliary constraints. *Backtrack point; all possible conflict resolution constraints need to be considered.*

2 **Consistency Check (Optional).** If the partial plan is inconsistent (i.e., it has no safe ground linearisations), fail. Else continue.

3 **Recursive Invocation.** Call *Refine-Plan* on the refined plan.

Figure 3.5: Generalised plan-space planning algorithm (after [Kambhampati et al 95])

Definition 3.8 (All-sol) *Given a partial plan \mathcal{P} , return success only when all ground linearisations of the plan are:*

- *safe, and*
- *satisfy all the PTC's.*

A ground linearisation G is said to satisfy a PTC $\langle c@t \rangle$ if there exists a step t' before t (t' could be t_0), such that the operator $ST(t')$ has an effect c and for every step t'' between t' and t in G , $ST(t'')$ does not delete c .

According to [Kambhampati *et al* 95, Page 189], ‘the solution constructors used by most existing planners correspond to some implementation of *all-sol*.’

3.4.4 Applying the Framework to Classical Planning Systems

The reward of setting up this formalism is its modularity; it now becomes possible to describe individual planning architectures by specifying the way in which they behave at each point in the architecture. Kambhampati’s summary of how existing plan-space planners can be categorised is shown in Figure 3.6 on the next page.

From this we can see, for instance, that while some planners implement a solution constructor function by explicitly checking the conditions of an MTC, others use protection strategies to guarantee that solutions exist. These protection strategies are facilitated by bookkeeping steps, of which the main varieties are interval protection and contributor protection.

Protection strategies are easy to characterise using the auxiliary constraints introduced earlier. For example, consider the act of adding a step t to a plan to provide some goal p for another step t' . *Interval protection* of this goal would then involve the posting of a constraint that prevents any other steps from deleting p between t and t' . This is simply modelled by adding the IPC $\langle t, p, t' \rangle$. *Contributor protection*, on the other hand, further specifies that p cannot be *added* by any other steps between t and t' . This is modelled by adding the extra constraint $\langle t, \neg p, t' \rangle$

Other important differences are revealed in the possible tractability refinements used by planners. Since checking whether a plan has a safe ground linearisation requires ex-

Planner	Soln. Constructor	Goal Selection	Bookkeeping	Tractability Refinements
TWEAK [Chapman 87]	MTC based $O(n^4)$	MTC-based $O(n^4)$	None	None
UA [Minton <i>et al</i> 91]	MTC based $O(n^2)$	MTC-based $O(n^2)$	None	Unambiguous ordering
NONLIN [Tate 77]	MTC (Q&A) based	Arbitrary $O(1)$	Interval & Goal Protection	Conflict Resolution
TOCL [Barret & Weld 93]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor Protection	Total Ordering
PEDESTAL [McDermott 91]	Protection based $O(1)$	Arbitrary $O(1)$	Interval Protection	Total Ordering
SNLP [McAllester & Rosenblitt 91] UCPOP [Penberthy & Weld 92]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor Protection	Conflict Resolution
MP, MP-I [Kambhampati 92]	Protection based	Arbitrary	(Multi) contributor protection	Conflict resolution

Figure 3.6: Characterisation of plan-space planners as instantiations of *Refine-Plan*. The n used in the complexity figures is the number of steps in the partial plan. ([Kambhampati *et al* 95, Page 26])



amining a potentially infinite number of ground linearisations, the task is intractable. One way to avoid explicitly enumerating all of these, however, is to utilise *pre-ordering refinements*. [Kambhampati *et al* 95, Page 196] describes *total ordering* and *unambiguous ordering* as follows:

Total ordering orders every pair of steps in the plan, while unambiguous ordering orders a pair of steps only when one of the steps has an effect c , and the other step either negates c or needs c as a pre-condition (implying that two steps *may* interact). Both of them guarantee that in the refinements produced by them, either all ground linearisations will be safe or none will be (in the case of total ordering, this holds vacuously true since the plan has only one linearisation). Thus, consistency can be checked in polynomial time by examining any one ground linearisation.

Another possibility here is *conflict resolution*, which attempts to replace the auxiliary constraints of a partial plan with equivalent ordering and binding constraints. When all conflicts are resolved in this way, the resulting partial plans will ‘have the property that all their ground linearisations are safe...Thus, checking the partial plan consistency will amount to checking for the existence of ground linearisations’ [Kambhampati *et al* 95, Page 33]. For more complete descriptions of this, and other details on classical planning, readers are referred to [Kambhampati *et al* 95].

3.5 Summary

We have looked at classical planning systems and presented a formalism within which they can be compared and contrasted. As well as providing a unified and coherent background, this formalism will be useful later in the thesis when we come to describe some of the techniques used by FINESSE for planning in the game of Bridge.

Chapter 4

The Bridge Search Space Size

Factorials were someone's attempt to make math *look* exciting.

— STEVEN WRIGHT

In Chapter 2 we examined some of the techniques that designers of computer Bridge programs have used to control search (rule-based systems, tactics, decomposition into sub-problems, and exhaustive search on double-dummy sub-problems). The FINESSE system uses what could be described as the knowledge-based approach of decomposing the problem into card combinations and then restricting the possible actions to a set of tactics. In this chapter we present a justification for this by analysing the typical search space size for a Bridge card play problem. This analysis is based on the calculation of the number of possible *legal play sequences* (lps's) in a slightly generalised version of Bridge, where the deck of cards contains four suits of N cards each, so that each player starts the game holding N cards. We use our results to assess the relative potential of knowledge-based and brute-force approaches to the game, and also speculate on the future of Bridge playing programs in general.

We proceed as follows. In §4.1 we establish some very basic upper and lower bounds on the search space size. §4.2 then introduces the concept of *shape*, which we make use of in §4.3 to improve our lower bounds both in the situation where all the cards are visible and then under the more normal assumption that two hands are unseen. We also interpret these results, suggesting that Bridge may be more amenable to solution by knowledge-based rather than brute-force approaches. In §4.4 we then examine the special case of double-dummy Bridge, and §4.5 gives a summary.

4.1 Preliminary Estimates

We begin by making some simple observations which allow rough upper and lower bounds on the number of lps's to be established. In fact, an upper bound is easily produced by just noticing that the highest possible number of cards that a player may choose from on any one round is equal to the total number of cards he holds. Thus, if we can construct a situation where each player has this freedom on every round (no matter what the distribution of the unseen cards), we can be sure that this will produce the largest possible search space.

Such a situation does in fact occur when one player is dealt a complete suit of cards. Assuming that this player retains the lead throughout the play, the final three players on any trick will always be void in the suit which is led, allowing their contribution to the trick to be made from a free choice amongst their remaining cards. Since the player who leads is also free to play any card from his complete suit, there are N^4 ways to play the first trick, $(N - 1)^4$ ways to play the second, $(N - 2)^4$ for the third, etc. An upper bound on the number of lps's is therefore $(N!)^4$, which when N is 13 as in actual Bridge, gives approximately 1.50×10^{39} .

A lower bound of $N!$, (or 6.28×10^9 for Bridge) can be produced by a similar argument, this time simply ignoring any choices that the final three players on a trick may have, and taking the free choice of the first player to be the sole contribution to the number of possibilities.

Values for the number of lps's when two hands are unknown can be produced by assuming that these bounds on the number of lps's hold in each of the possible game states reached by dividing the unknown cards between the two unseen hands. Since there are ${}^{2N}C_N$ ways of carrying out this distribution, the upper and lower bounds in Bridge are 1.56×10^{46} and 6.48×10^{16} respectively.

Being general bounds on the number of lps's in an arbitrary Bridge card play problem, these bounds are rather loose. A more realistic solution would capture the tendency of the search space to increase with the skewness of the card distribution. In particular it seems that the lower bound on the number of lps's severely underestimates the contribution of particularly uneven distributions. Below, we develop a more realistic

formula for the number of lps's in a given situation, making use of the Bridge notion of the *shape* of a hand.

4.2 Shape

The *shape* of a hand is simply the number of cards it contains in each of the four suits ♠, ♥, ♦, and ♣. When discussing Bridge problems, it is common to describe a player's hand by talking about its shape, since this allows a basic amount of information to be conveyed quickly and efficiently, without resorting to naming each card individually. The following hand, for example, can be easily characterised as having a 4-4-4-1 shape.

♠ A K Q J
♥ 9 8 7 5
♦ A Q 6 2
♣ 4

We can make use of this concept of shape to derive a new lower bound on the number of lps's possible in a given situation. To do this we first look at the number of choices open to one player whose hand is of shape $S_1 - S_2 - S_3 - S_4$.

Figure 4.1 shows the choices that can be made by this player during the play of a hand. The root node of the tree describes the initial shape, and the nodes at the second level represent the possible shapes that may be reached after one trick. Since advance knowledge of the suit which will be led at the start of each trick is impossible, four arcs emanate from the root node, the first representing the transition that takes place on a ♠ lead, the second on a ♥ lead, the third on a ♦, and the fourth on a ♣. The labels attached to these arcs indicate the number of choices open to the player in each of these circumstances.

If we make the conservative assumption that there is no point in the game at which the player wins the lead, the cards that are chosen by the other players to begin each trick will determine a path through the tree of alternatives. The number of choices made by the player in following this path will therefore be the multiple of all the labels attached to the path's branches.

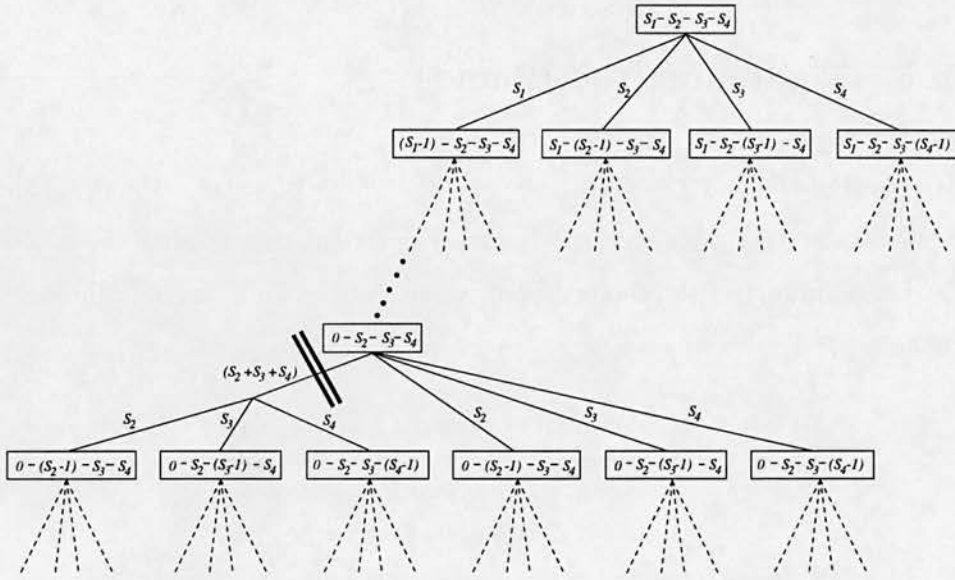


Figure 4.1: Choices and resulting shapes for a single player

It is fairly easy to show that $S_1!S_2!S_3!S_4!$ is a lower bound on the value of this number of choices. If the path through the tree involves no nodes at which the player is void in the suit which is led, then the number of choices he makes is obviously $S_1!S_2!S_3!S_4!$, since at the end of the game he has no remaining cards, and has therefore made S_1 plays in the spade suit, S_2 in the heart suit, etc. If, on the other hand, tricks are initiated by cards of a suit in which the player is void, we can still show that $S_1!S_2!S_3!S_4!$ is a lower bound for the number of eventual choices. For example, consider the node in Figure 4.1 at which the player's shape is $0-S_2-S_3-S_4$. If a spade is led at this point, there are $S_2 + S_3 + S_4$ alternative choices for the player, since he may choose any of his S_2 cards in the heart suit, or one of his S_3 cards in the diamond suit, or one of his S_4 cards in the club suit. However, notice that each of the shapes that can be reached by following this branch is duplicated on one of the branches representing the other possible leads. Since the contributions of these other branches to the total number of choices are just S_2 , S_3 and S_4 respectively, we can prune the spade branch, effectively replacing its $S_2 + S_3 + S_4$ choices by *either* S_2 , S_3 or S_4 . By repeating this pruning argument at all nodes where the player has zero cards in some suit, the only remaining paths will be those in which there are $S_1!S_2!S_3!S_4!$ choices, and all the pruned paths

will have at least this number of choices.

4.3 Tightening the Bounds

To use the above argument on shapes to produce a better estimate of the number of lps's in a given Bridge game, it is simply sufficient to note that the number of lps's is just the product of the choices faced by each of the four players. Thus, if we define s_{ij} as follows:

$$s_{ij} = \begin{cases} \text{the number of cards held by player } i \text{ in suit } j, \\ \text{where } i = 1 \dots 4 \text{ are the players' hands, and} \\ \quad j = 1 \text{ is the } \spadesuit \text{ suit, } j = 2 \text{ is } \heartsuit, \\ \quad j = 3 \text{ is } \diamondsuit, \text{ and } j = 4 \text{ is } \clubsuit. \end{cases}$$

the number of possible lps's is given by the formula:

$$\prod_{i,j=1}^4 s_{ij}!, \quad (4.1)$$

which in Bridge achieves its maximum (1.50×10^{39}) when all the players hold one complete suit, and its minimum (7.22×10^{14}) when each player holds 4 cards in one suit and three in all the others. Comparing these figures with the bounds from §4.1 (1.50×10^{39} and 6.23×10^9) we can see that although (4.1) provides us with the same upper bound, its lower bound is significantly tighter. We therefore now go on to utilise this new bound in the general situation where there is incomplete information.

4.3.1 Factoring in the Number of Possible Deals

Using (4.1) as the lower bound on the number of lps's when the shape of all four hands is known, we can attempt to calculate the minimum number of lps's in the normal situation where just two hands are known. Let us assume that the hands which are visible are described by the values S_{ij} , for $i = 1, 2$ and $j = 1 \dots 4$. If we let \mathcal{S} denote the set of possible shapes which respect this constraint and $number(s)$ denote the number of card distributions which have shape $s \in \mathcal{S}$, the formula for the number of lps's can be written as:

$$\sum_{s \in \mathcal{S}} \text{number}(s) \prod_{i,j=1}^4 s_{ij}!.$$

Now, $\text{number}(s)$ is just the number of ways of distributing the cards not present in the visible hands to produce the shape s . If we write the number of cards remaining in suit k as the constant $N_k = (N - S_{1k} - S_{2k})$, the number of ways of assigning s_{3k} cards to player three in suit k can be written as:

$${}^{N_k}C_{s_{3k}} = \frac{N_k!}{s_{3k}!(N_k - s_{3k})} = \frac{N_k!}{s_{3k}!s_{4k}!}.$$

The number of ways of producing a given overall shape is therefore:

$$\text{number}(s) = \prod_{k=1}^4 \frac{N_k!}{s_{3k}!s_{4k}!},$$

and the total number of possible lps's becomes:

$$\begin{aligned} & \sum_{s \in \mathcal{S}} \left(\prod_{k=1}^4 \frac{N_k!}{s_{3k}!s_{4k}!} \prod_{i,j=1}^4 s_{ij}! \right) \\ &= \sum_{s \in \mathcal{S}} N_1!N_2!N_3!N_4! \prod_{i=1}^2 \prod_{j=1}^4 s_{ij}! \\ &= N_1!N_2!N_3!N_4! \prod_{i=1}^2 \prod_{j=1}^4 S_{ij}! |\mathcal{S}|. \end{aligned} \tag{4.2}$$

where $|\mathcal{S}|$ denotes the size of the set \mathcal{S} , (*i.e.*, the total number of possible shapes, given that two hands are fixed).

To produce a numerical value from this formula we require a way of calculating the value of $|\mathcal{S}|$. Below, we therefore derive an expression for this, by first examining the number of shapes that an individual hand may take when being dealt cards from a complete deck, and then considering how many of these shapes become impossible when the number of available cards is reduced to N_k in each suit.

4.3.2 The Number of Possible Shapes of a Single Hand

Figure 4.2 shows how N cards may be dealt to a single player from a 4-suited pack of cards containing N cards per suit. In the first suit, the player may have any number of cards from 0 to N (represented by the arcs branching from the root node). Dealing a certain number of cards in this suit, however, restricts the number which can then be dealt in the second suit. If no cards are dealt in the first suit, there may obviously again be up to N cards in the second suit, but if one card is dealt in the first suit, the maximum number of possible cards in the second suit drops to $N - 1$ (since otherwise, the hand would contain more than N cards). The branches at the second level of the tree in Figure 4.2 show how this pattern continues, until the maximum number of cards in the second suit finally decreases to 0 when the number of cards in the first suit is N . A similar pattern occurs on the third level of the tree (representing the number of cards possible in the third suit), with the number of alternatives decreasing as the sum of the cards already allocated becomes larger. Finally, at the bottom level of the tree, the number of cards in the final suit is uniquely determined by the restriction that the player's hand must contain N cards in total.

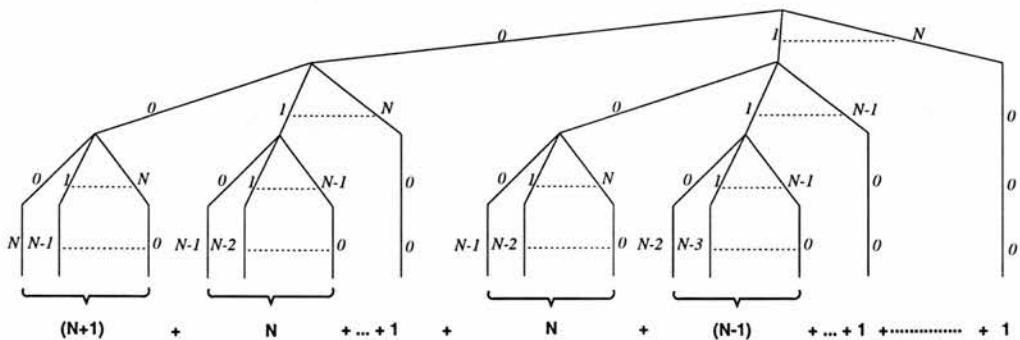


Figure 4.2: Number of ways to produce shapes for a single hand

Each possible path through the tree, then, describes one of the shapes that a single player's hand may take (for example, the leftmost path represents the shape 0-0-0- N), and summing all the possible paths will give us the total number of possible shapes.

At the base of the tree, branches with a common root at the second level have been grouped together and annotated with a term describing the number of branches in the

group. From this it is clear that the number of paths with n cards in the first suit is:

$$\sum_{i=1}^{N-n+1} i.$$

The total number of paths is therefore:

$$\begin{aligned} & \sum_{j=1}^{N+1} \sum_{i=1}^j i \\ &= \sum_{j=1}^{N+1} \frac{j(j+1)}{2} \\ &= \frac{(N+1)(N+2)(N+3)}{6}. \end{aligned} \tag{4.3}$$

In Bridge, where N is 13, the number of possible shapes for an individual hand is therefore 560.

4.3.3 The Number of Possible Shapes of The Unseen Hands

How does (4.3) help us to find the size of the set \mathcal{S} ? Well, notice that when assigning the $2N$ unknown cards to the unseen hands, specifying the shape of just one hand will also completely specify the other. The number of possible shapes that the unseen hands may take is therefore equivalent to the number of shapes for a single hand when dealing from a pack with N_k cards in each suit, where $N_1, N_2, N_3, N_4 \leq N$, and $N_1 + N_2 + N_3 + N_4 = 2N$. The possible shapes which may be produced in this scenario are therefore a proper subset of the possible shapes in Figure 4.2, since the reduction of the number of available cards will make some of the branches in this tree impossible. For example, if $N_1 < N$, all the root-node branches from $N_1 + 1$ up to N will no longer be possible. We will call the number of branches pruned by reducing N_k in this way, $D(N_k)$. Its value is given by the following:

$$\begin{aligned} D(N_k) &= \sum_{j=1}^{N-N_k} \sum_{i=1}^j i \\ &= \frac{1}{2} \sum_{j=1}^{N-N_k} j(j+1) \end{aligned}$$

$$= \frac{(N - N_k)(N - N_k + 1)(N - N_k + 2)}{6}. \quad (4.4)$$

The total number of paths which are pruned by reducing the number of cards in each suit to N_k , however, is not simply $\sum_{k=1}^4 D(N_k)$, since this expression may count more than once any shape that is pruned when there are insufficient cards available in more than one suit. (For example, the shape 6-6-1-0 will contribute twice to this sum whenever both N_1 and N_2 are less than 6). To rectify this, we will define $I(N_k, N_l)$ to be the number of shapes which become impossible both when suit k is reduced to N_k cards and when suit l is reduced to N_l cards (in other words, the number of shapes with more than N_l cards in suit l that are counted in the calculation of $D(N_k)$). The value of $I(N_k, N_l)$ can be calculated as follows:

$$\begin{aligned} I(N_k, N_l) &= \sum_{i=1}^{N-N_k-N_l-1} (N - N_k - N_l - i)i \\ &= (N - N_k - N_l) \sum_{i=1}^{N-N_k-N_l-1} i - \sum_{i=1}^{N-N_k-N_l-1} i^2 \\ &= \begin{cases} \frac{(N-N_k-N_l-1)(N-N_k-N_l)(N-N_k-N_l+1)}{6} & \text{if } N_k + N_l \leq N - 1 \\ 0 & \text{otherwise.} \end{cases} \quad (4.5) \end{aligned}$$

It is possible to show that with $2N$ cards remaining there are no shapes which can be pruned three times, so that the formula for the total number of shapes is:

$$|\mathcal{S}| = \frac{(N+1)(N+2)(N+3)}{6} - \sum_{k=1}^4 D(N_k) + \sum_{k=1}^4 \sum_{l=k+1}^4 I(N_k, N_l). \quad (4.6)$$

To see this, consider the conditions that must hold if a shape $s'_1 - s'_2 - s'_3 - s'_4$ is to be pruned three times. Let us say that the three suits in which $N_k < s'_k$ are those given by $k = 1, 2, 3$. The inequality $N_1 + N_2 + N_3 \leq s'_1 + s'_2 + s'_3 - 3$ must therefore hold. Since $s'_1 + s'_2 + s'_3$ must in turn be less than or equal to N , we can write $N_1 + N_2 + N_3 \leq N - 3$. In our situation of distributing the final $2N$ cards between the opponents, having $N - 3$ cards available in three suits would require there to be $N + 3$ in the final suit. This, of course, cannot occur, since each suit only contains N cards when complete, and therefore the situation where a shape is pruned three times is not possible.

4.3.4 The Incomplete Information Search Space in Bridge

Given a particular Bridge position, the shape information on the visible hands can be substituted into (4.2), (4.4), (4.5), and (4.6) to find the lower bound on the number of lps's. Using a Prolog program, we did this for all the possible compatible shapes which two visible hands could take. Also, for each of these assignments of shapes, we calculated the number of ways in which a standard pack of cards could be dealt to fit the constraints. In Figure 4.3, we present the results of this program in the form of a graph where each point represents the number of deals for which a specific lower bound was returned by our equations. As one might expect, the most common deals (at the right hand side of the graph) have relatively low search spaces (the lowest value is 9.91×10^{22}), and as the value of the lower bound increases the number of deals for which this bound holds also goes down. It is interesting to note that the relationship between the lower bound and the number of deals having this bound appears to be roughly linear (with a negative coefficient).

Interpretation

We can use the graph of Figure 4.3 to speculate on the likely prospects of computer Bridge programs. For instance, consider Figure 4.4 on Page 64 in which Allis 'very crudely' details the number of search space positions in the games which appeared in the 2nd Computer Games Olympiad.

Note that this table is concerned with the number of *positions* in each search space. In many games, the number of lps's isn't finite, because of loops, so it makes sense to talk instead about the number of search space positions as a measure of difficulty. In Bridge, however, there is a finite number of legal play sequences, because each sequence is only 52 moves long. Since we can count the number of legal play sequences, we have a lower bound on the number of nodes in the actual game tree. We can argue that this is close to the number of positions in the search space because firstly the exponential nature of the tree means that the number of nodes at the final level make the largest contribution to the total and secondly, there are some 'identical' nodes in the space, which reduce the total number of search space positions.

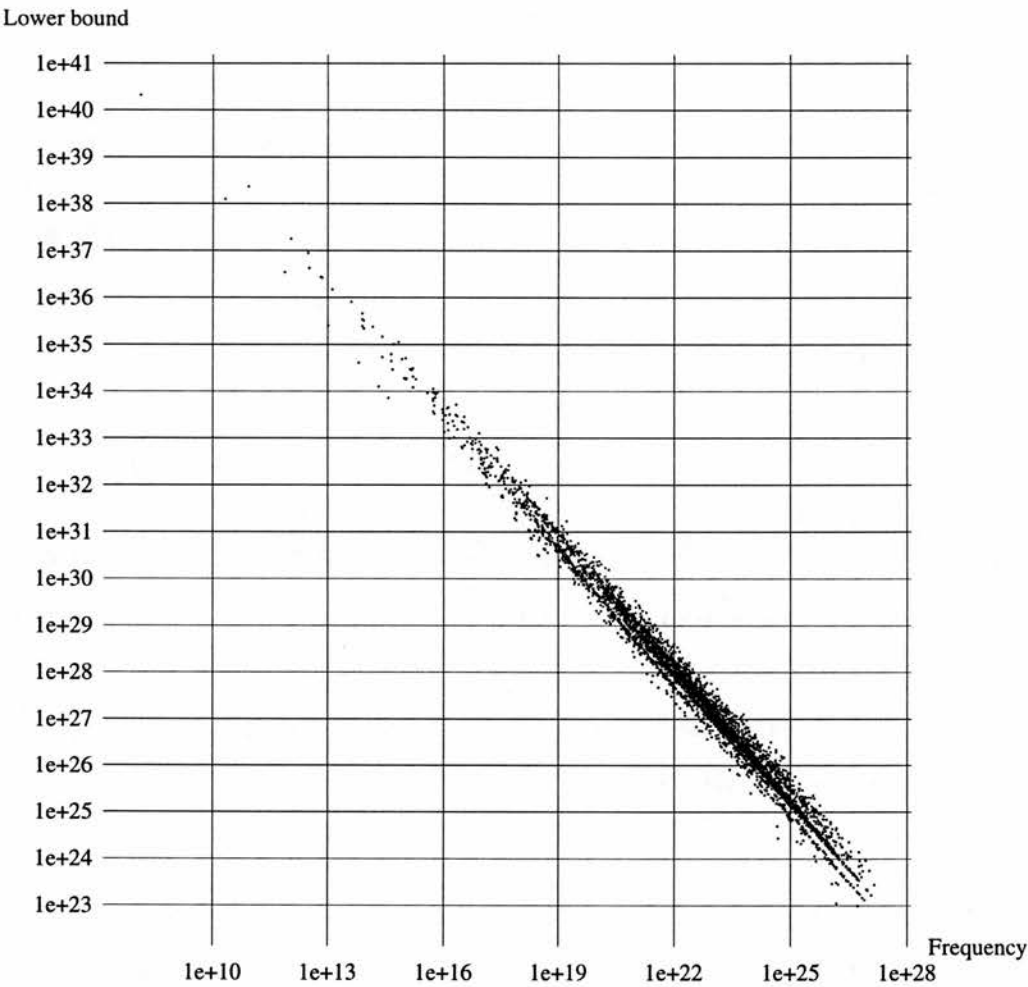


Figure 4.3: Log-scaled graph of search space lower bound against frequency

Game	$^{10}\log$ of positions in search space
Nine Men's Morris	11
Awari	12
Connect-Four	14
Backgammon	19
Checkers (8×8)	20
Qubic	30
Othello	30
Bridge	30
Draughts (10×10)	35
Go (9×9)	35
Chinese Chess	45
Chess	50
Go-Moku	105
Renju	105
Scrabble	150
Go	170

Figure 4.4: Estimated search space complexities [Allis *et al* 91, Page 236]

Given this, we can see that our graph of Figure 4.3 is in broad agreement with Allis' estimated figure for Bridge of 10^{30} . However, it adds the dimension of visually demonstrating that the number of possibilities depends upon the starting position and also that lower search space sizes are more likely. We can therefore broadly reinforce the claim which Allis makes based on his figures that Bridge playing programs will at best be performing at Grand Master strength by the year 2000.

One remarkable fact about Bridge which these figures highlight is the extraordinary ability of humans to play the game. Although we would not expect a computer to be able to search the size of space discussed above, this is precisely the ability displayed by human players! In Checkers, with a comparable search space size, no player (human or otherwise) can sit down at the table at the beginning of a game and announce that they have decided upon their best moves through to the end. However, this is in fact what human declarers do in Bridge, forming and comparing complete strategies in a matter of tens of seconds before playing to trick one.

Explaining this human ability to effectively search such a large space so quickly may well be the key to understanding the poor performance of computer Bridge programs

to date. One possibility is that human declarers achieve this by employing the common problem-solving strategy of splitting the initially complex problem into several smaller sub-problems which are then worked on independently. This is the approach taken by FINESSE, which separately tackles the sub-problems of the play of the card combinations in each individual suit, as discussed in Chapter 2.

We suggest that in Bridge it is the presence of a natural decomposition into sub-problems, combined with the ability of humans to learn techniques for reasoning about the dependencies between sub-plans, that makes computers appear to be poor at Bridge. In games like chess and checkers, where such decompositions are difficult to identify, both humans and computers achieve limited lookahead (the computer by sheer brute force, humans by recognising promising patterns and areas of the search space), and the gap in performance is now being closed. In Bridge, however, limited lookahead is comprehensively outclassed by any human opponent who can project play through to the crucial end stages of the play by considering how plans for each suit can be pieced together. Indeed, faced with such a human opponent, it is questionable whether brute-force lookahead over the number of lps's indicated above will ever prove to be competitive. For instance, recall from §2.5 the analysis of the team designing the CHINOOK checkers playing program [Schaeffer *et al* 93a]. They cite an instance from a match against the World Champion Marion Tinsley, where CHINOOK played its 10th move and on the basis of a 17-ply search claimed a small advantage. However, the game was lost in 35 moves and afterwards Tinsley revealed that 'on move 12 he had seen to the end of the game and knew he was going to win'. The best estimate of the CHINOOK team was that this feat involved 'selective searches of over 60 ply'. In a game such as Bridge with a comparable search space size and human opponents that are regularly able to project the play through to its conclusion, such occurrences will surely be even more likely.

A (presumably knowledge-based) system capable of forming complete global plans by handling multiple sub-plans and reasoning about their interactions may fare better, but it should be pointed out that this approach poses serious complications. A large proportion of AI planning research has concentrated on simple blocks-world-like domains [Fikes *et al* 72], [Sacerdoti 74], with no concept of an opposition, or of actions

with multiple possible outcomes [Wilkins 88]. Even in these simple domains, the ‘multiple goal plan optimisation problem’ (of which declarer play is an example) is NP-hard [Nau *et al* 91]. Given this, the lack of progress on a domain with a search space as large as that of Bridge is hardly surprising.

4.4 Double-dummy Bridge

In Chapter 2, we noted that some authors have tried to simplify the task of automating Bridge card play by solving the simpler double-dummy situation, where all the players reveal their cards to each other.

One obvious way to assess the feasibility of creating a fast double-dummy solver is to produce some measure of the search space size typically involved in such a problem. It turns out that this question has been tackled by other authors with arguments about the number of lps’s that may occur. Berlekamp, for example, claims that ‘for most deals, the size of the complete tree of legal moves is about 10^{20} ’ [Berlekamp 63]. Levy, however, argues that the number is ‘somewhat less, nearer to 10^{16} ’, presenting an argument which leads to an estimate of $10^{15.7}$ [Levy 89]. Levy goes on to point out two types of errors in his method of estimation, one of which tends to over-estimate the answer and the other to under-estimate. Confusingly, he then states his belief that ‘these errors would cancel each other out to the extent that $10^{15.7}$ is more likely to be an upper bound than it is to be a lower bound’. Clearly, $10^{15.7}$ cannot be an upper bound, since we have already seen from (4.1) that the number can be as high as 1.50×10^{39} . However, such a large lower bound is of course very unlikely to occur. More useful than a simple lower bound, then, would be a lower bound on the *expected* number of lps’s in a randomly generated double-dummy problem. Below, we adapt our previous analysis to produce an answer to this question.

4.4.1 The Expected Number of Legal Play Sequences

Let us again consider a four-suited pack of cards with N cards per suit. The total number of possible deals is $(4N)!/(N!)^4$. Let us consider one particular deal in which the cards are distributed with shape s_{ij} for $i, j = 1 \dots 4$. If we look at just the cards

in one suit, say spades, then the number of ways to deal player 1's holding is:

$${}^N C_{s_{11}} = \frac{N!}{s_{11}!(N - s_{11})!},$$

and the number of ways of dealing the remaining $(N - s_{11})$ cards in the suit to player 2 is:

$${}^{N-s_{11}} C_{s_{21}} = \frac{(N - s_{11})!}{s_{21}!(N - s_{11} - s_{21})!}.$$

The number of ways of dealing the spades to the remaining players can be expressed in the same way, and multiplying these expressions together to find the number of ways in which the suit can assume a given shape produces:

$$\frac{N!}{s_{11}!s_{21}!s_{31}!s_{41}!}.$$

The number of deals which can assume the shape s_{ij} for $i, j = 1 \dots 4$, then, is:

$$\frac{(N!)^4}{\prod_{i,j=1}^4 s_{ij}!}.$$

(4.1) already gives us a lower bound on the number of lps's for a deal with a given shape, so if we let \mathcal{S}' be the set of possible shapes that an entire hand may take, a lower bound on the *expected* number of lps's in a double-dummy problem may be derived as follows:

$$\begin{aligned} \text{Lower Bound} &= \sum_{s \in \mathcal{S}'} \text{prob}(s) \prod_{i,j=1}^4 s_{ij}! \\ &= \sum_{s \in \mathcal{S}'} \frac{\text{number of deals of shape } s}{\text{total number of deals}} \prod_{i,j=1}^4 s_{ij}! \\ &= \sum_{s \in \mathcal{S}'} \frac{(N!)^4 / \prod_{i,j=1}^4 s_{ij}!}{(4N)! / (N!)^4} \prod_{i,j=1}^4 s_{ij}! \\ &= \sum_{s \in \mathcal{S}'} \frac{(N!)^8}{(4N)!} \\ &= \frac{(N!)^8}{(4N)!} |\mathcal{S}'|. \end{aligned} \tag{4.7}$$

To find the total number of possible shapes, we resorted to a brute-force programming solution and, for $N = 13$, produced the value 3.75×10^7 . (4.7) therefore gives us a lower bound on the expected size of a double-dummy problem in regular Bridge of 1.05×10^{18} . This falls midway between Berlekamp's estimate and Levy's 'upper' bound.

4.4.2 Interpretation

The lower bound which (4.7) places on the expected size of a double-dummy search problem at least suggests that Levy's estimate of the number of double-dummy searches possible per second may need to be revised downward by a factor of a hundred or so. This rather undermines his argument that the use of various heuristics and the alpha-beta algorithm with killer heuristics could reduce the $10^{15.7}$ nodes of each double-dummy search space to a practically manageable number.

However, it is, of course, difficult to formally analyse the effects of some of the search reduction techniques that may be applicable. For instance, Levy argues that 'the real improvements in search' in Bridge will come from 'the nature of the problem', since some of the most skewed distributions where the search space is apparently highest are also the easiest in practice to play. (For example, when a player holds a complete suit of cards, his choice of play on each round is actually immaterial, as he can play the cards in any order without affecting the result.)

Unfortunately, the possible benefits of such considerations also remain unproven in practice. To date, Berlekamp [Berlekamp 63] and Wheen [Wheen 89] have produced programs that work in a matter of minutes for end-game situations where each player holds a reduced number of cards, but which take at best in the order of hours for complete hands. Ginsberg, however, has recently reported some success, claiming in postings to the Internet Usenet group `rec.games.bridge` a reduction in the average branching factor for the number of possible options on each move to 1.27, thus enabling a single hand to be analysed in 'about thirty seconds'. It is unfortunate that, at the time of writing, very few details about Ginsberg's system are available, making close analysis impossible.

4.5 Summary

We have looked at the search space size in Bridge, initially establishing simple upper and lower bounds, and then producing a detailed analysis utilising the concept of the shape of a hand. Our analysis suggests that Bridge programs using a brute-force approach may find it difficult to compete with human players.

We also examined the special case of double-dummy Bridge, establishing a lower bound which was higher than that suggested by some other authors. We will return to the notion of using fast double-dummy solvers as the basis for a Bridge program in Chapter 6, where we will discover that it suffers from rather more serious shortcomings.

Chapter 5

Proof-planning: Solving Independent Goals using Tactics and Methods

‘Now,’ said Rabbit, ‘this is a Search, and I’ve Organised it -’

‘Done what to it?’ said Pooh.

‘Organised it. Which means - well, it’s what you do to a Search, when you don’t all look in the same place at once.’

— A. A. MILNE
The House at Pooh Corner

In the beginner’s mind there are many possibilities, but in the expert’s there are few.

— SHUNRYU SUZUKI

In this chapter, we begin the description of the FINESSE system itself, starting with the *pre-planner* and the *planner* modules. A basic framework for these modules was developed previously as an MSc project [Frank 91], producing legal play sequences for single-suit Bridge problems. In the course of incorporating this into a larger system capable of planning complete games of Bridge, however, a substantial amount of modification became necessary. This chapter describes both the surviving portions of the original and these modifications.

The main feature to have survived intact is the notion, borrowed from *proof-planning*, of using *tactics* to specify the possible operators that can be applied at any stage of the plan. The features that have been either enhanced or newly introduced fall broadly

into the following categories:

- **Representation.** Simplifications were made to the representation of the plan structure itself, and to the representation of the preconditions under which each tactic could be used.
- **Efficiency.** The performance of the planning algorithm was improved, both as a result of the above representation changes and also by the introduction of a ‘history’ mechanism for reducing duplication of effort.
- **Interface.** We introduced a front-end capable of generating a graphical rendition of the planner’s output, and also of tracing the planner’s operation. This proved a significant aid in the debugging of the system.

A further addition to the original system was an algorithm for identifying the *best* choices from among the play sequences generated by the planner. However, it transpires that this problem is very subtle and we will therefore describe it later, in a chapter of its own. Here we just concentrate on the basic problem of identifying and representing the possible legal play sequences.

We proceed as follows. In §5.1 we introduce the proof-planning paradigm and motivate its application to the game of Bridge. In §5.2 we then identify the commonly occurring tactics for the declarer in single-suit Bridge situations, and in §5.3 we look at how the opponents may respond to such tactics. §5.4 examines how the preconditions of Bridge tactics can be represented as methods, and §5.5 then shows how these methods can be used in a planning algorithm. We then go on in §5.6 to examine the interface to the planning system, and also, in §5.7, to look at some efficiency issues. Finally, §5.8 gives a summary.

5.1 Proof-Planning

The technique of *proof-planning* has been developed by the Mathematical Reasoning Group at Edinburgh University to find proofs for mathematical theorems [Bundy 88, Bundy 91]. The basic task in proving a given theorem is to construct a proof tree linking the theorem to some known axioms, via rules of inference, such as the decomposing

of a hypothesis, or the cutting in of a new assumption. This task may be approached by starting with either the theorem to be proven and working backwards, or with the axioms and working forwards. In Edinburgh it is tackled as a decomposition process, so in our diagram of Figure 5.1 we place the theorem to be proven at the top of the tree. Employing the planning terminology introduced in Chapter 3, proof-planning involves a state-space search and produces a plan in a state-space representation. Notice that any particular rule of inference in this space may give rise to a number of subgoals, all of which must be proven if the inference rule is to be successfully applied. This is represented in the diagram by the use of circular ‘AND’ nodes.

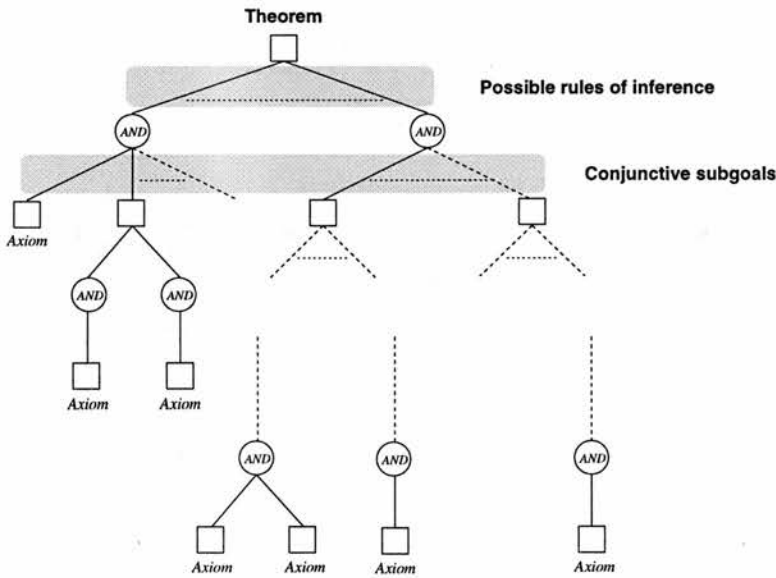


Figure 5.1: The state-space search in theorem-proving

Rather than search this actual space, which is potentially infinite in size, the proof-planning paradigm restricts the operators available at any point to a specified set of possibilities, which are encoded as programs that construct proofs in the spirit of LCF *tactics* [Gordon *et al* 79]. The preconditions and postconditions of these tactics are in turn specified in a *meta-language* to form *methods*, so that planning consists of combining these methods by reasoning about their specifications. In practice, the *meta-level* search space formed by reasoning with the method specifications focuses the search within areas where it is more likely to succeed. The average branching factor at the nodes of this space is reduced by many orders of magnitude over that of the

object-level space, where proof construction consists of applying the primitive inference rules.

As we saw in the previous chapter, the search space in Bridge card play, like that of mathematical theorem-proving, is very large — certainly too large to be exhaustively searched by humans. This, however, has not prevented experts from becoming remarkably adept at navigating these spaces, building up a large variety of techniques that they can draw upon to successfully manoeuvre within them. The success of proof planning by declaratively capturing such human problem solving strategies in theorem-proving [Bundy *et al* 91] was the main motivation for wanting to apply the same techniques to Bridge.

This motivation was further reinforced by the resemblance between the search spaces of theorem-proving and of Bridge declarer play: whereas theorem-proving requires repeated applications of rules of inference to a theorem until axioms are reached, the course of a game of Bridge proceeds with the playing of individual cards until none remain. This analogy is illustrated in Figure 5.2.

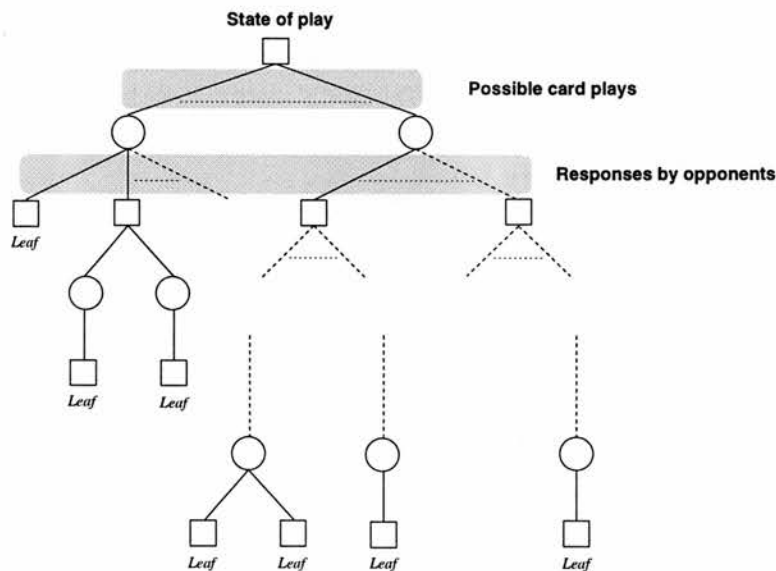


Figure 5.2: The state-space search in Bridge card play

The square nodes now correspond to states of play rather than theorems to be proven, and the branches at the circular nodes represent the possible responses which the

opponents may make to the declarer's possible actions. Thus, Bridge card play and theorem proving share a basic similarity in form.

5.2 Bridge Tactics

In Bridge, the most basic operation on the state of a game is the play of a single card. Rather than planning the play by considering all the possible combinations of these cards, however, any human declarer will look for commonly occurring patterns that fit the current situation. As suggested above, the size of the search space in Bridge card play can be reduced by specifying such patterns as *tactics* and restricting the declarer's options at each stage of planning/play to a pre-determined set of such possibilities. Here, we review the tactics for No Trumps, single-suit situations which were identified as part of the original FINESSE project. We illustrate each tactic with a few single-suit card combinations, assuming in each case that South is the declarer, so that North's cards are laid on the table and are played by South. Also, we will use an 'x' to represent an arbitrary low card. It should be noted that the majority of the text describing these tactics is taken, with minor edits and corrections, from [Frank 91].

5.2.1 Cashing

Cashing is probably the simplest, and most intuitive of all the tactics used by human Bridge players. Whenever the declarer holds any *masters* — cards that are higher than any held by the opponents — in a particular suit, he may *cash* one by leading a low card towards it, or by leading the master itself. In a No Trumps situation, an attempt to cash a master will always succeed, since it cannot be ruffed. As examples, consider the situations of Figure 5.3 on the next page.

The first of these examples is simple: there is one master and one low card, so the obvious (and only) choice is to cash the Ace. In the second example, declarer has two masters, so a decision has to be made: which card should be cashed first, the Ace or the King? It is clear that attempting to cash the Ace first would be a mistake, since the King would also have to be played on the same trick, leaving declarer with no masters on the subsequent round. Cashing the King is therefore the correct play. In

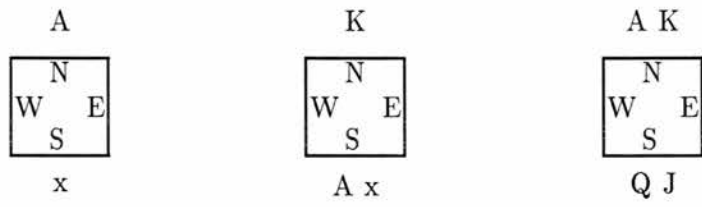


Figure 5.3: Card combinations in which declarer can cash winners

the third example, declarer has four masters but can only cash two of them due to a shortage of length.

Most Bridge books are full of ‘explanations’ of card play techniques in the same ‘example-driven’ vein as those presented above. With the possible exception of computer learning systems however, this kind of ‘knowledge’ is not in a very suitable form for incorporation into computer programs. One of the major challenges of the original FINESSE system was to convert all the expertise contained in book examples into rules that could be expressed as essentially declarative Prolog code. For example, the first situation above would suggest the rule ‘If you have a master, cash it’. Considering the second situation would lead to a refinement such as ‘Any master can be cashed provided the opposite hand has a lower card which is not a master (or is void)’. Then the final situation calls for the relaxation that ‘If declarer’s longest hand in the suit is shorter than the number of masters, a master may be cashed even if the lower cards in the opposite hand are also all masters’.

This task of deciding the rules that govern the applicability of each tactic, however, is properly the job of the *methods*. We will describe these more fully, and give some examples, in §5.4. Here we just concentrate on identifying the possible play patterns.

5.2.2 Finessing

Finessing situations occur when the declarer is trying to develop extra tricks, and are based on the elementary principle of card play that the best results can be obtained by forcing an opponent to play ahead of you. A simple illustration is that of Figure 5.4. Here, it is obvious that the declarer will win one certain trick (in a No Trumps situation)



Figure 5.4: A simple card combination allowing a finesse

when he plays the Ace. However, if he were to play this card first, his only chance of making *two* tricks would be if one of the defenders held the singleton King, so that it would fall when the Ace was played. Thus, the cashing of the Ace succeeds in winning two tricks in only two of the possible 2^{10} distributions of the outstanding cards.

There is, however, another way in which two tricks could be made in the suit: if the King is held by West, and if West can be compelled to play before the North hand. To take advantage of this possibility, the proper procedure is to enter the South hand and lead the two. If West now plays the King, it is beaten by the Ace, and a second trick can be won on a later round with the Queen. If West keeps the King, the trick can be won by playing the Queen from the North hand. Of course, if East holds the King, dummy's Queen will lose this trick, but in this situation declarer's only chance of making two tricks would have been to hope that East's King was singleton, so that it would fall when the Ace was cashed. Clearly, the chances of success offered by inserting the Queen are better than this, since it brings in two tricks whenever West holds the King, a 50/50 chance.

Again, however, the Bridge literature on this type of play is long on examples and short on concrete specifications. As part of the original FINESSE system, the wide variety of finesses found in the literature was divided into a number of broadly distinct types. Perhaps surprisingly, it was found that just four categories were sufficient to describe all the examples which were encountered. We describe these below, after establishing the following terminology:

- **Finesse Card.** A card with which declarer tries to win a trick, even though it is not a master (in the above example, this was the Queen). Since the finesse card

is never a master, at least one of the defenders will hold a card that can beat it. Any such card is a card that is being *finessed against*.

- **Lead Card.** A card which declarer leads in order to initiate the finesse.
- **Cover Card.** A card in the third hand capable of beating at least one of the cards that is being finessed against. Unlike finesse cards and lead cards, possession of a cover card is not a necessary prerequisite for all the types of finesse.

Type 1 finesses

In this type of finesse (probably the most common of the four) the finesse card is accompanied by a cover card in the same hand (a combination often referred to as a *tenace*). Any card from the hand opposite the tenace that is lower than the finesse card can be used as a lead card. The example of Figure 5.4 was a Type 1 finesse, in which declarer attempted to win an extra trick with the Queen by leading a low card towards it from the South hand.

Type 2 finesses

In Type 2 finesses, the lead card is the finesse card, and the hand opposite must contain a cover card, as in the situations of Figure 5.5.

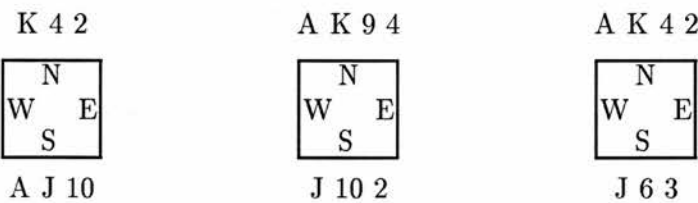


Figure 5.5: Card combinations allowing Type 2 finesses

In the first two of these examples the Jack or the 10 may be used as the finesse/lead card. In each case, the card that is finessed against is the Queen, and the hand opposite the Jack and 10 contains a cover card for this.

In the third example, however, the lead of the Jack would be improper, because even if West holds the Queen he will presumably cover and now one of the opponents must win a trick with the 10. This leads us to propose the restriction that declarer must hold at least one card immediately above or below a Type 2 finesse/lead card.

Type 3 finesses

In this type of finesse, no cover card is required. It is best illustrated by an example (see Figure 5.6).



Figure 5.6: A card combination allowing a Type 3 finesse

Here, declarer would obviously like to win tricks with both the King and the Queen. This is possible, but can only be done if West holds the Ace, and if he is compelled to play before the North hand. To achieve this, the declarer leads a small card from the South hand. If West plays the Ace, declarer's troubles are over. If he plays small, declarer wins with the Queen and enters the South hand again to repeat the process. Note that if declarer had led the King from the North hand, he could only have taken one trick.

Type 4 finesses

In the situation of Figure 5.7 on the following page, it is possible to take three tricks if East holds the Queen. A play that achieves this is to cash the King on the first round and then execute a Type 1 finesse of the Jack on the second, by leading a small card from the North hand.

Sometimes, however, there may be extra information (for example, from the bidding)



Figure 5.7: A more complicated finessing situation

which indicates that the Queen is with West. Is it now hopeless to make three tricks? No, there is one chance, and that is that East has the 10. [Goren 86, Page 342] describes the correct play:

...execute what is known as the backward finesse, leading the Jack first. Notice that you must not lead the Ace first. When the Jack is covered by the Queen, you win with the King and return a low card, finessing the 9, with the hope that East holds the 10. If West holds both, there is nothing you can do about it.

This lead of the Jack is different to those of the first three finessing tactics. Since the Jack plays the role of both the finesse card and the lead card, the play looks very similar to a Type 2 finesse. In Type 2 finesses, however, the lead card must come from a sequence, which is not the case with the Jack here. Under normal circumstances, one would expect the first defender to cover such a lead, removing the chance of any gain for the declarer. The backward finesse gets around this problem by creating a new finessing situation against the opposite defender when the first defender covers.

The possibility of leading a lone finesse card was explicitly guarded against in Type 2 finesses, since it usually produces no gain. To remedy this situation, we introduce a new tactic, the Type 4 finesse, which generates just such a lead. A Type 4 finesse is defined as being applicable whenever a lone finesse card can be led to create a (Type 1 or Type 2) finesse situation on the following round.

It is important to realise that this tactic does *not* correspond to the play of two finesses. It merely generates a lead for a finessing round which would be overlooked by the first three finessing tactics. The finesse which is applicable on the second round will have to be discovered by the planner, when it examines the possible states resulting from a

Type 4 finesse. To see why it is expedient to represent the tactic in this way, consider the situation of Figure 5.8, in which a Type 4 finesse of the Jack is followed not by a Type 1 finesse but by a Type 2 finesse of the 8.



Figure 5.8: A further example of a backward finesse

5.2.3 Ducking

Ducking entails playing low from both hands on one round of a suit. Declarer gains no immediate benefit from this exercise, but instead hopes to increase his chances of creating extra tricks on the following rounds. An example of a situation in which a declarer might consider ducking is shown in Figure 5.9.



Figure 5.9: A card combination in which ducking may profit

With this holding, declarer is certain to win one trick with the Ace (given a No Trumps situation), but ideally he would like to win with the Queen as well. As we have seen already then, the crucial play, is a (Type 1) finesse of the Queen against West — a play with a 50/50 chance of success, since the declarer will win two tricks whenever the King is with West. These odds can be improved slightly, however, if the declarer ducks one round in the suit before attempting the finesse. If East’s only card in the suit is the King, he will be forced to play it, allowing the Ace and Queen to be cashed on the

following rounds. The advantage gained by this round of ducking is rather small (in this case, gaining an extra trick one under just one of the 2^8 possible distributions of the outstanding cards in the suit) but a declarer would possibly attempt the play if it was inevitable that one trick had to be lost in this suit¹ in any case.

In addition to being used in an attempt to remove the defence's high cards, ducking may also be employed as a means of retaining *control* in a suit. This is illustrated by the example of Figure 5.10 which we take from [Goren 86, Page 341].



The North hand has no other entries than Spades, and your object is to win five tricks. It can be seen that regardless of how the adverse cards are distributed one trick must be lost in any event. Therefore a trick should be conceded to the opponents at once. Now our only hope is for West to have the King. The finesse is taken the next time, and if the King is in the West hand, dummy's Spades will all be good. It is true that if the finesse loses you will succeed in taking no spade tricks instead of one, but it was your only hope and it was well worth spending an extra trick to try it.

Figure 5.10: A card combination illustrating ducking [Goren 86, Page 341]

5.2.4 Top Sequences

Declarer may find himself in the fortunate position of having enough high cards to play one on each trick. If this is the case, there is obviously no point in using the finessing or ducking tactics, and Figure 5.11 presents some situations in which even cashing is not possible.

In the first situation, the appropriate action is to either lead one of the K, Q or J or to lead a small card towards them. Since all the cards that constitute the declarer's top

¹ Note that another way to account for the singleton King is to cash Ace and then execute a Type 3 finesse of the Queen. This plan would have the added advantage of not necessarily giving up one trick to the defence.

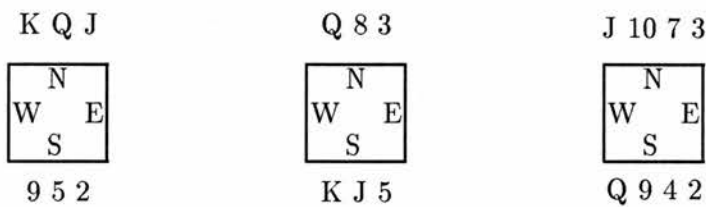


Figure 5.11: Card combinations where declarer has high cards but no winners

sequence are contained in the same hand, the effect of playing a round to the Q or the J is exactly the same as that of playing to the K, so there is really just one meaningful course of action. This is not the case in the other situations, however, where declarer’s top sequence is split between the two hands. In the second situation, for example, if the defence decline to play their Ace, playing one of the top cards on a trick can leave the lead in *either* the North or the South hand. Since declarer can play a high card from the sequence on each round of the suit, playing either the King or the Jack will essentially be equivalent. Therefore there should be just *two* possible courses of action in this situation: play the Queen or play one of the King/Jack.

These situations are not covered by any of the tactics we have identified so far, so we will introduce another, calling it the *sequence* tactic. In general, whenever sequence tactics are applicable to a game state, there will be either exactly one or exactly two valid instantiations: one if the entire sequence is contained in one hand, or two if the sequence is split between the two hands.

5.2.5 Summary of the Tactic Set

In summary, then, seven distinct tactics for single-suit problems were identified in the original FINESSE system, and retained during this research. This set of tactics has been sufficient to describe the optimal lines of play for all the single-suit situations to which the system has so far been applied. They are represented by the following Prolog predicates:

- 1–4. `finesse(Type, Player, Card, Suit)` — `Type` represents the type of finesse being used; `Player` is the defender being finessed; `Card` and `Suit` specify the

finesse card.

- 5. `cash(Card, Suit)` — represents a trick on which declarer plays the card specified by `Card` and `Suit` from one hand, and plays a low card (or throws away a card from another suit) in the other.
- 6. `duck(Suit)` — represents a trick on which declarer plays low from both his hands.
- 7. `sequence(Card, Suit)` — represents a trick on which declarer plays the card specified by `Card` and `Suit` (which must come from a sequence of length 2 or more) from one hand, and plays low (or throws away a card from another suit) in the other.

Having examined the possible plays by the declarer in single-suit situations, we now turn our attention to the complementary problem of representing the possible responses that can be made by the opposition, again drawing on [Frank 91].

5.3 Representing the Defenders' Plays

Given the declarer's holding in a particular suit and a choice of tactic, what are the possible plays by the defence when the tactic is executed? In theory, any combination of the outstanding cards may appear on the trick. For example, in the situation of Figure 5.12 there are seven outstanding cards in the suit: the A, K, J, 8, 7, 6 and 3.



Figure 5.12: Example card combination with seven outstanding cards

If each defender plays a card from the suit when a tactic is executed, there will be seven choices for the first defender, and six for the second (42 choices in total). In addition, there is the chance that one of the defenders may be void in the suit, and will

therefore have to *discard* a card from another suit. Even ignoring the actual identity of these discards, this creates another fourteen possibilities (seven in which the first defender is void, and seven in which the second defender is void), making a grand total of 56 possible responses by the defence. Admittedly, some of these possibilities may be somewhat remote (*e.g.*, both the Ace and the King being played) but all must be catered for if we are to produce a fully predictive plan.

In general, if the defence hold a total of n cards, the number of possible plays to a single trick is given by the formula $n(n-1)+2n$, which simplifies to $n(n+1)$. Reducing this branching factor is an important goal for the FINESSE system, which it achieves by merging together plays which are in some sense ‘equivalent’, as we describe below.

5.3.1 Card Sequences

Opportunities for reducing the branching caused by the defenders’ possible moves arise in any situation where the playing of different cards lead to a similar new game state. One example of such situations is when the defenders hold a *sequence* of cards.

For example, in Figure 5.12 the defenders hold two sequences of length 1 (the J and the 3), one sequence of length 2 (the A K) and one sequence of length 3 (the 8 7 6). We make the simplification that the play of one card from each of these sequences is completely equivalent to the play of another. In practice, this is not strictly true, since conventional meanings can be attached to playing certain cards and used to pass *signals* to other players. We will return to this issue in more detail in §6.2.4. For now, we will use the simplification to allow some of the defence’s possible plays to be identified together. For example, the case where East plays the 3 and West plays the Ace can be merged with the case where East plays the 3 and West plays the King to form a single situation where East still plays the 3 but West plays *either* the Ace or King. In general, if the outstanding cards form n_1 sequences of length 1 and n_2 sequences of length greater than 1, the number of distinct possible responses by the defence to a single tactic (including the cases where one defender is void) is expressed by:

$$n_2(n_1 + n_2) + n_1(n_1 + n_2 - 1) + 2(n_1 + n_2)$$

$$= (n_1 + n_2 + 2)(n_1 + n_2) - n_1. \quad (5.1)$$

In the example of Figure 5.12, where $n_1 = n_2 = 2$, the number of possible combinations therefore decreases from the original 56 to 22 — a significant saving.

5.3.2 Critical Cards

It could be argued that the efficiency improvement achieved by identifying sequences above is a trifle fortuitous, as the outstanding cards happen to contain two sequences of length greater than 1. For example, had the seven outstanding cards been the A, K, J, 8, 6, 4, and 2, (*i.e.*, $n_1 = 5$ and $n_2 = 1$) there would still have been 43 possible branches. However, there is a further technique that we can call to our aid, based on the approach, common among human Bridge players in such situations, of ignoring the distinction between all the small cards held by the defence, and classifying them all as ‘low’. Take, for example, Goren’s description of the situation in Figure 5.12 [Goren 86, Page 311]

If West holds both the Ace and the King, you can win one trick if you lead from the South hand..... However, inasmuch as it is improbable that West has both the Ace and the King, the better procedure is to wish that he holds the Jack. A small card is led from the South hand, and when West plays low North follows with the 9, hoping that this will drive out either the King or the Ace. The South hand is entered and another small card is led, and since West is known to hold the Jack one trick must be built up.

The Ace, King, and Jack are identified as the ‘important’ cards, and the play of any of the others is described as ‘playing low’. We will use the term *critical* cards to describe the high cards still held by the defence that are considered to be ‘important’. In FINESSE, these cards are identified by a *pre-planner* module which then passes them to the planner. This module survives largely intact from the original system and is described in [Frank 91].

By classifying the outstanding cards as either critical cards or low cards, the search space branching factor due to the defenders’ play can be more consistently reduced. For instance, in the example of Figure 5.12, the critical cards are the Ace, King and Jack. Each defender may therefore choose between the following possibilities:

- The Ace, King (a sequence of length 2),
- The Jack (a sequence of length 1),
- A low card (a sequence of length 4), or
- A card from another suit.

Under this categorisation, n_1 is 1 and n_2 is 2, so we can see from (5.1) that the number of possible branches is now just 14. In later chapters, we will see that this categorisation also helps in the generation of textual explanations.

5.4 Methods

In Edinburgh's theorem-proving system (CLAM, [Bundy *et al* 90b, van Harmelen 89]), methods are the basic stuff that make up proof plans. They are specifications of tactics, which are procedures that execute a (large) number of proof steps as a single operation. CLAM's methods are structures with 6 'slots':

1. A *name-slot*, giving the method its name, and specifying the arguments to the method.
2. An *input-slot*, specifying the object-level formula to which the method is applicable.
3. A *preconditions-slot*, specifying conditions that must be true for the method to be applicable.
4. A *postconditions-slot*, specifying conditions that will be true after the method has applied successfully.
5. An *output-slot*, specifying the object-level formulae that will be produced as subgoals when the method has applied successfully.
6. A *tactic-slot*, giving the name of the tactic for which this method is a specification.

These structures are represented as a Prolog `method/6` term. Each of the slots corresponds to an argument of the `method/6` term, in the order listed above. The general form of a `method/6` term is shown in Figure 5.13.

```

method(name(...Args...),      % name slot: Prolog term
        H==>G,                % input slot: sequent
        [...Preconditions...], % preconditions-slot: list of conjuncts
        [...Postconditions...], % postconditions-slot: list of conjuncts
        [...Outputs...],      % output slot: list of sequents
        tactic(...Args...))    % tactic slot: Prolog term
).
```

Figure 5.13: The general form of a `method/6` term [van Harmelen 89]

In these methods, the post-conditions and preconditions are combined together, forming an object which in essence resembles a STRIPS type operator [Fikes *et al* 72]. Unfortunately this kind of representation was found to be ill-suited to planning in Bridge, as the results of applying a Bridge tactic depend on the particular response chosen by the opponents. For example, in the Type 1 finesse tactic described in §5.2.2, the card played by the third hand depends on the card chosen by West: if West plays the King, declarer plays the Ace; otherwise declarer plays the Queen. This is an example of the *context-dependency* of actions we discussed in Chapter 3.

FINESSE's approach to dealing with context-dependency is to use a database of rules which, for each tactic, allows the declarer's cards to be chosen and played, taking into account the cards chosen by the defence. This deduction of context-dependent effects is very similar to the way SIPE [Wilkins 84] uses domain rules to alleviate problems in operator representation caused by the STRIPS assumption. A consequence of using such rules is that the amount of information which has to be stored in FINESSE's methods is reduced. In fact, the number of 'slots' required by FINESSE is just three; the post-conditions and output-slots are no longer needed (since they are deduced by the domain rules), and a further slot is removed since the name-slot and the tactic slot always contain the same terms. Figure 5.14 on the following page shows the general form of the method predicates used by the original FINESSE system.

The significant information in these methods is the specification of the preconditions. These are written in a meta-language which survives largely intact from the original

```

method(name(...Args...),      % tactic/name slot: Prolog term
      State,                  % input slot: game state
      [...Preconditions...]) % preconditions-slot: list of conjuncts
).
```

Figure 5.14: The general form of a `method/3` term in the original FINESSE [Frank 91]

system. The types of properties which can be described by this language are suggested in Figure 5.15, which gives an example of the preconditions of a tactic in plain English.

<div style="text-align: center;"> <p>A J 9</p> <table border="1" style="margin: auto;"> <tr><td></td><td>N</td><td></td></tr> <tr><td>W</td><td></td><td>E</td></tr> <tr><td></td><td>S</td><td></td></tr> </table> <p>3 4</p> </div>		N		W		E		S		<p>Preconditions:</p> <ul style="list-style-type: none"> - Declarer must have at least one <i>loser</i>² in the suit. - Declarer has a sequence of any length beneath a critical card. - Declarer has a card that can beat the critical card, in the same hand as a card from the above sequence. - The opposite hand has a low card to lead toward the tenace. - The player being finessed is not known to be void in the finesse suit.
	N									
W		E								
	S									

Figure 5.15: Preconditions of the Type 1 finesse [Frank 91, page 38]

In the course of this research the major change to FINESSE’s methods was the re-writing of these pre-conditions to remove Prolog cuts. This was an unpleasant feature of the original system, as it compromised the declarative nature of the method specifications. The re-writing of the meta-language to remove the need for such occurrences resulted in a clearer representation, and also paved the way for a change in the actual representation of the methods. Given the reduction in the number of method slots to just three, a more natural representation is to directly utilise Prolog clauses of the form:

`applicable(State, Tactic) :- PreConditions.`

Notice that this representation has clear efficiency advantages over the original, due to the checking of preconditions being conducted by the Prolog interpreter as part of an attempt to satisfy an `applicable(+State, ?Tactic)` goal, instead of by a user-defined procedure examining a *list* of preconditions. However, a planning algorithm

² See the Glossary for a definition of the Bridge notion of losers.

using backtracking to produce all the possible legal play sequences would have been unable to utilise such a representation if Prolog cuts were permitted as part of the language describing preconditions.

5.5 Finesse's Planning Algorithm

The tactics we have introduced in this chapter do not specify complete strategies to be followed for any particular card combination, but only continuations for the next trick. The task of examining all the possible legal play sequences is performed by a planning algorithm which constructs a tree of tactics.

In the original version of FINESSE, the planning algorithm was modelled on those used by CLAM in the domain of mathematical theorem-proving. At any node at which more than one tactic was applicable, a choice was made according to a heuristic ordering. This algorithm therefore could only return *subtrees* of the possible space; trees in which each branching point controlled by the declarer had only one branch. The remaining possible subtrees could also be returned on backtracking, but the entire search space was never explicitly represented. In CLAM, a substantial amount of progress has been made in specifying progressively higher-level tactics and methods to direct this type of search algorithm to the best, or at least good, solutions without the need for large amounts of backtracking. Bridge, however, presents new demands on planning that are not encountered in mathematics, the most significant being the need to cope with an opposition. Since cards played may not be retracted once plan execution has begun, the value of a Bridge planning system hinges on its ability to identify the most beneficial alternatives *before acting* in any particular situation. This is not an issue in mathematical proofs, since one may always backtrack from failed branches until a successful proof is found. Furthermore, as information on the distribution of the opponents' cards is initially unknown, probabilistic information must be used in Bridge to choose among possible plans. It is therefore important for us to examine as many play sequences as possible before selecting the one which is believed to be the *best*. Since the reduction in search space size achieved by restricting the declarer's options at any point to a set of methods proves sufficient to allow the whole search space to be examined (relative to the tactic/method set), we therefore implemented a new plan-

ning system which could produce an explicit representation of the entire search space. Such plans are then passed to a separate module which identifies the branch selections which would be optimal (we discuss this in the next chapter).

To form a plan, the planning algorithm first calls an `applicable(+State, -Tactic)` goal to find all the `Tactics` applicable to the initial `State`. The possible responses by the defence are then generated, and the post-conditions determined. This process is continued recursively for the resulting states until no tactics are applicable to any of the leaf nodes in the tree. Thus, planning involves a form of heuristically constrained state space search, using the declarer's initial holdings as the starting state, and working forwards until reaching a state to which no tactics apply.

5.5.1 A Planning Example

We will demonstrate this planning procedure with the help of an example. Figure 5.16 shows how the plan for the game state where North holds the A, Q and South holds the 2 is built up. There are two tactics applicable to this state: the cashing of the Ace and a Type 1 finesse of the Queen against West.

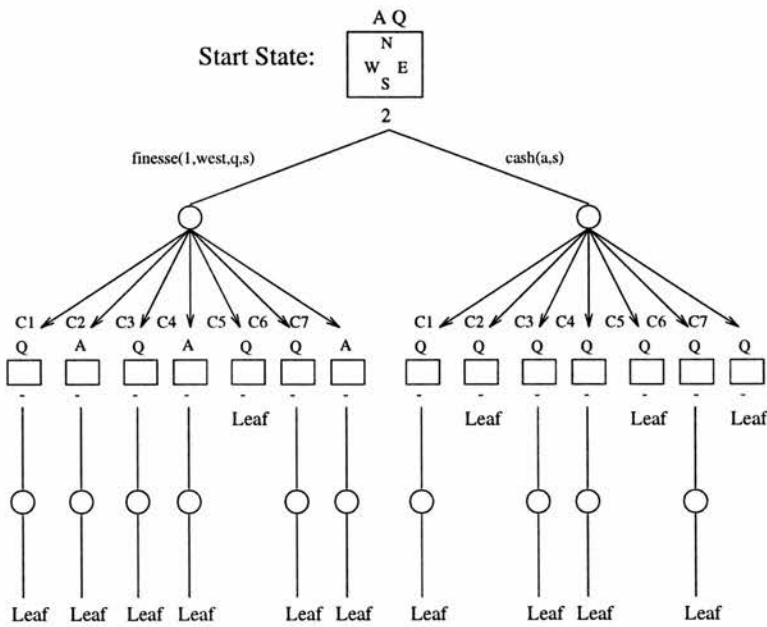


Figure 5.16: A planning example

The responses to these initial tactics are represented in FINESSE by the seven possible plays possibilities enumerated in Figure 5.17. Here, ‘discard’ refers to playing a card from a different suit (*i.e.*, the defender is void), and it is assumed that the King is the only critical card, and that there is no trump suit.

Code	Card played by West	Card played by East
C1	King	discard
C2	low	discard
C3	discard	King
C4	low	King
C5	discard	low
C6	King	low
C7	low	low

Figure 5.17: Key for Figure 5.16

Removing the cards played by each hand leads to new states, from which planning continues on the next round. In the case of the cash tactic, all possible branches lead to a state where North holds the Queen and South is void. For the finesse tactic, however, the card played by North is determined by West’s play after the 2 is led: if the King appears then declarer covers with the Ace; otherwise, the finesse is attempted by playing the Queen.

In some of the states at the second level, declarer’s remaining high card will be a master, so he will be able to continue the play by cashing this card. This is represented in the figure by the further level of play on which each branch represents a cash tactic. For each of these tactics, the cards played by the defence are immaterial, as declarer will not be playing any further tricks in the suit. Therefore, there is only one branch at each of the circle nodes. The declarer wins two tricks on any path that is two tricks long, and one trick otherwise (assuming no trump suit).

The tree of Figure 5.16 is represented in FINESSE using a Prolog term of the form: `functor(Node, Labels, Trees)`, where:

- `functor` is an arbitrary atom (`tree` or `plan`, for example),
- `Trees` is a list of subtrees which form the trees rooted on the daughters of the

current `Node`,

- `Labels` is a list of terms describing the branches leading from the current `Node` to each of the subtrees in `Trees`, and
- leaf nodes are represented as `functor(Node, [], [])`.

5.6 Interface Issues

Interpreting and visualising the rather large data structures produced by the planning algorithm described above was found to be a non-trivial task. To facilitate easier debugging and to increase the inspectability of the output, a Prolog package capable of producing a graphical rendition of tree data structures was written.³ This package enabled FINESSE to display its plans in windows such as that shown in Figure 5.18 on the following page, which depicts the system's output for the simple planning example considered in the previous section. (Note that the cards A, K, Q, and J, are represented by the numbers 14, 13, 12, and 11, respectively. Also, this display uses the string 'Max' to label nodes where it is the declarer's turn to play, and 'Min' to label those where it is the turn of the defence. These names are motivated by the well-established literature on 2-player games, which we will examine in detail in the following chapter).

In order to further enhance the utility of this interface, the ability to examine the planning state at any node of the tree was introduced. For this purpose, the planning algorithm was modified to store data on the important planning information in each `Node` of the plan representation. To facilitate this, the data contained in this argument is represented as a list of `field:data` pairs, where the `field` describes the meaning of the `data` entry it is paired with. Examples of the possible `fields` utilised by the planner are:

- `label` — the name of the node (*i.e.*, Min, Max, or Leaf).
- `state` — the current game state (which specifies the North the cards, the South

³ This package was designed for portability, and has since been incorporated into further systems by other researchers. For example, in [Vasconcelos 95] it is used to visually realise Prolog programming techniques.

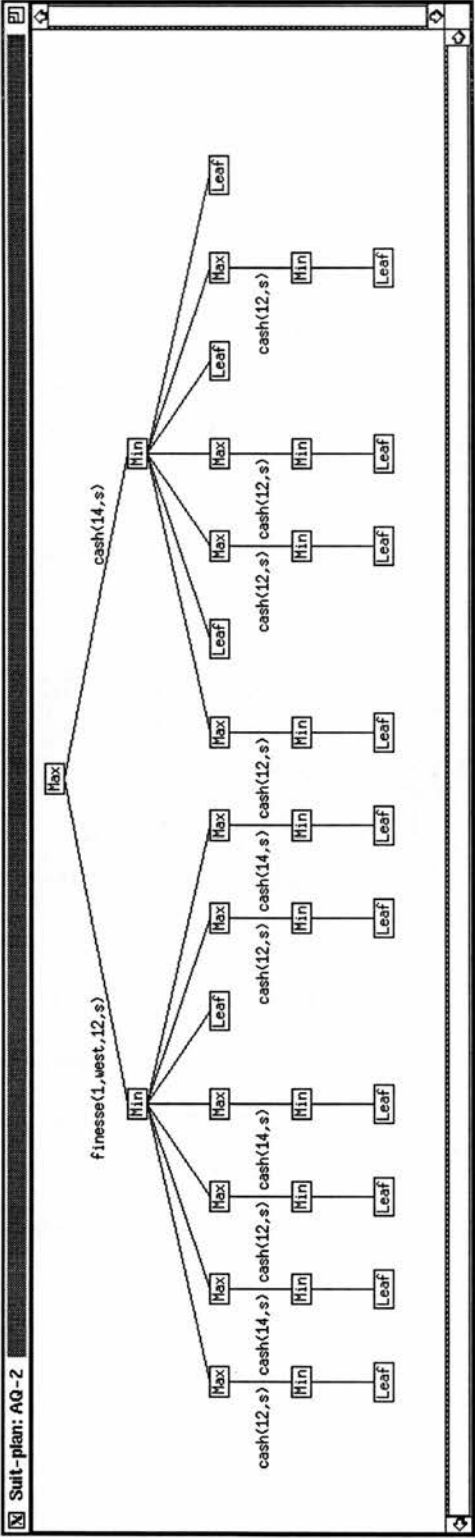


Figure 5.18: Example graphical display of plan data structure

cards, the outstanding cards, the critical cards, and two flags indicating whether East or West are known to be void in the suit).

- tactics — the applicable tactics (only relevant to Max nodes).
- branches — the possible card plays (only relevant to Min nodes).

At any node in the plan, the values of these data items can be examined by simply clicking on the node to produce a pop-up window, as shown in Figure 5.19 on the next page. The pop-up window for the root node of the tree shows how the data associated with the state field is displayed graphically by a picture of the game state.

5.6.1 Tracing the Planner

When upgrading FINESSE to produce an explicit representation of the entire search space (relative to the method set), it was found that some problems for which the original planner was capable of producing partial trees now caused the new system to crash due to segmentation faults or failed memory allocations. This behaviour was attributed to either the actual plan formation process exceeding the memory limitations of the machine, or to the possible presence of looping. In order to diagnose the latter of these possibilities, some way of monitoring the planner's execution was necessary. Under the original implementation, this was not readily possible, so a modified planning predicate was implemented allowing a window to be opened onto the planning operations. Figure 5.20 on Page 96 shows an example of such a *trace* window monitoring the planning of the situation of the $AQ\spadesuit$ opposite the $2\spadesuit$.

Experimentation with this tracing facility, however, failed to locate any looping on problem examples. In fact, all the planning states being reached appeared to be legitimate, and breakdowns occurred at times when the planner appeared to be operating normally. The alternative explanation for FINESSE's problems — insufficient physical memory — was therefore examined. Reducing the amount of memory required by the planner led to the resolution of the problems with the system and to a significant increase in efficiency. This was achieved through the introduction of the notion of planning *histories*, which we describe below.

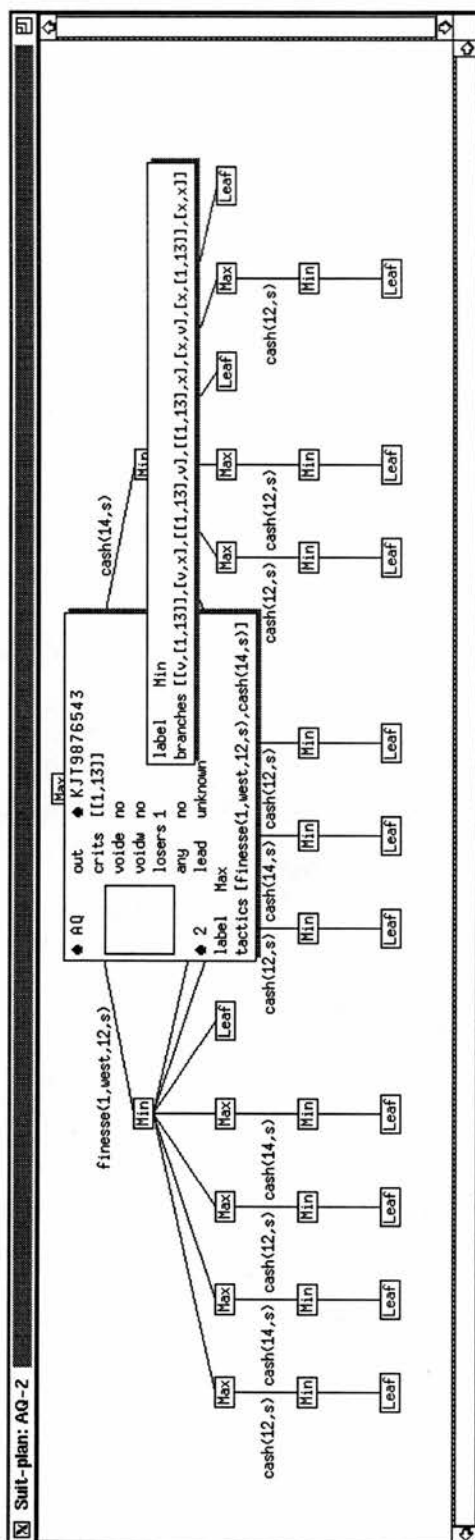


Figure 5.19: Example display of plan data structure showing pop-up windows

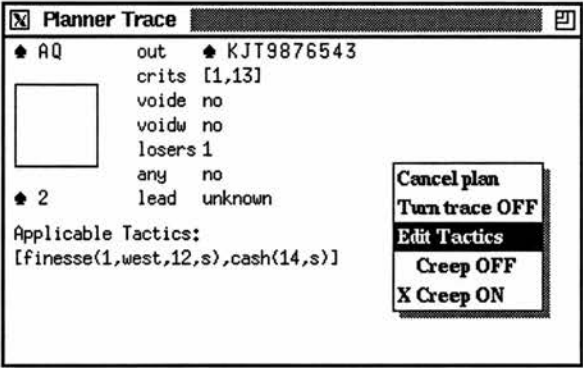


Figure 5.20: Example trace window showing menu options

5.7 Searching with Histories

In the process of constructing a Bridge plan, it transpires that the same planning state may be encountered a number of times. An automated Bridge planning system therefore runs the risk of wasting time on repeated calculation of identical sub-plans. For example, let us consider once more the case of the AQ opposite the 2. In this situation, FINESSE has to choose between the two possible tactics of finessing the Queen (Type 1) against East, or simply cashing the Ace. However, in examining the possible responses by the defenders to these tactics, repeated states may be encountered. For example, on any branch where West plays the King, both the cash and the finesse tactic will lead to the same state (since both tactics will play the 2 from the South hand and the Ace from the North hand in this situation). Similarly, if West is void, the two tactics will also lead to identical states (since FINESSE’s domain rules choose to select the cash card when the defender being finessed turns out to be void). Further, when cashing the Ace, the two branches where one defender plays the King and the other plays low will also be indistinguishable, since the declarer’s cards will always be Ace and the 2 irrespective of the cards chosen by the defence. This means that of the fourteen states which are produced on the second level of the plan, only five will be unique, as shown in Figure 5.21.

In a simple situation such as this, the duplication of the effort involved in planning each of the repeated states afresh is not particularly significant. However, since the

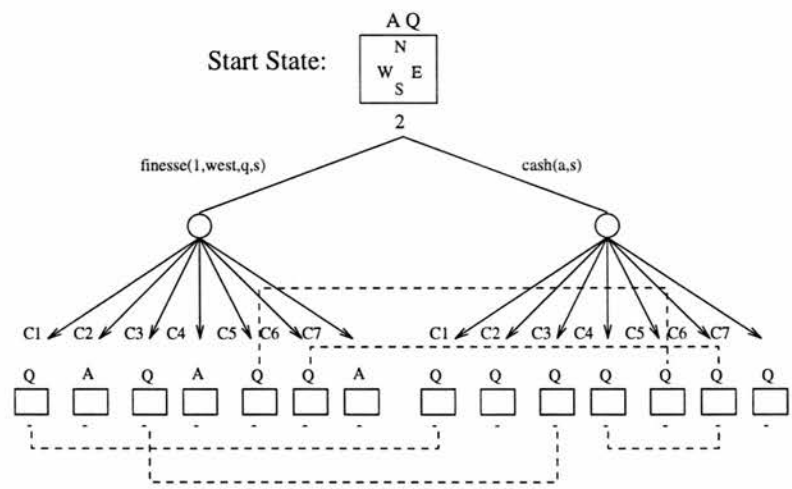


Figure 5.21: Example showing repeated states on the first planning level

size of a tree grows exponentially with the branching factor, the number of repetitions can be expected to be significantly larger in states with more applicable tactics.

In addition to the simple computation time that can be saved in not re-calculating repeated sub-plans, it is also possible that memory could be utilised more efficiently: constructing the finished data structure using pointers to shared sub-structures could avoid creating many copies of entire sub-trees. In §5.7.2 we examine how such efficiencies were achieved in FINESSE by the use of a *history list* mechanism. This very simple idea involves recording the plans that are generated for each distinct state encountered by the planner, so that when a state that has already been analysed is reached, the associated plan can be copied again rather than having to be re-generated from scratch.⁴ Before examining the results of this implementation, however, we introduce a short mathematical analysis to aid in their interpretation.

⁴ Note that this approach bears a resemblance to the dominance-based pruning technique of dynamic programming [Dreyfus & Law 77]. However, dynamic programming is more often used to find a *lowest-cost* solution to a search problem, utilising pruning to *remove* a node from a search space when the discovery of another, cheaper path to an identical node is made. The object of FINESSE's planner, on the other hand, is to generate *all the possible* play sequences in a given situation and previous planning results are therefore simply re-incorporated into the search tree when encountering repeated nodes.

5.7.1 General analysis

Utilising a history of states which have already been examined is simplified in the domain of Bridge by the knowledge that there is a measure of the states which strictly decreases as the tree is descended — the total number of cards remaining. Clearly, at any node in a tree the corresponding state will contain more cards than all the states of lower nodes, and less than those at higher nodes, since the descent of any branch involves the removal of cards. Thus, establishing whether a state has already been encountered requires only the checking of all the previous states at the same depth, and not the entire history.

In Prolog, support for hashing tables — the most efficient choice of data structure for implementing histories — is not readily available, so the basic list structure was adapted to the purpose. The history mechanism was therefore implemented by keeping a separate list of states and their calculated plans for each level of the tree. In general, under such a scheme, if the average branching factor between levels of a tree at which history checking is carried out is b , the number of extra comparisons made in checking the history list at level i when no matches are found is:

$$0 + 1 + \cdots + (b^i - 1) = \frac{(b^i - 1)(b^i - 2)}{2}. \quad (5.2)$$

If, however, we succeed in finding a plan from the history list for just one of the nodes at level i we will make the time saving of not having to expand all the nodes which would have been generated below it. (A memory saving is also produced if the algorithm is coded carefully, but here we will just consider the time savings, in order to produce a guide by which to measure the planner's performance.) In FINESSE, the main cost in expanding a node is due to the application of the method set to determine the applicable tactics. When the plan for a node at level i is found from the history list, then, the saving is on average $m \times b^{(d-i)}$, where m is the average cost of applying the method set to a single state, and d is the depth of the tree, which we assume to be uniform.

If the proportion of nodes at level i which are duplicated is some small fraction α_i , the saving made by using the history list is therefore

$$\alpha_i \cdot b^i \cdot m \cdot b^{(d-i)} = \alpha_i \cdot m \cdot b^d. \quad (5.3)$$

So, we have a saving at level i whenever the value of (5.3) is greater than the value of (5.2), or

$$\begin{aligned} \alpha_i \cdot m \cdot b^d &\geq \frac{b^{2i}}{2} \\ 2 \cdot \alpha_i \cdot m \cdot b^d &\geq b^{2i} \\ \ln(2 \cdot \alpha_i \cdot m) + d \cdot \ln(b) &\geq 2 \cdot i \cdot \ln(b). \end{aligned} \quad (5.4)$$

In FINESSE, the cost of applying the method set can be measured using simple experimentation, to be approximately 15 times as expensive as a simple unification check. Thus, if we approximate α_i to be $1/30$ (not unreasonable in FINESSE), (5.4) reduces to:

$$\begin{aligned} d \cdot \ln(b) &\geq 2 \cdot i \cdot \ln(b), \text{ or} \\ d/2 &\geq i. \end{aligned}$$

Therefore, it appears that keeping a history list is effective for any level above half the depth of the tree. This makes some kind of intuitive sense, since the cost of searching a large history list at lower levels of the tree could be expected to outweigh the benefits of the relatively small number of branches pruned.

5.7.2 Performance

A planner with the ability to carry history lists for any number of levels in a tree was implemented and tested on a number of problems. The bar charts in Figure 5.23 show some timings of this planner on the set of four test problems, of varying difficulty, shown in Figure 5.22. History lists are kept for each level of the plan (where it is the declarer's turn to play), so a maximum history depth of 0 is equivalent to using no history at all, and an increase in depth of 1 extends the history facility to the next

level. The height of the bar graphs represents the total computation time required for each test (so that a bar of height 10 seconds for a problem repeated 100 times indicates that each individual run took 0.1 seconds), and were produced on a Sun SPARCstation ELC.

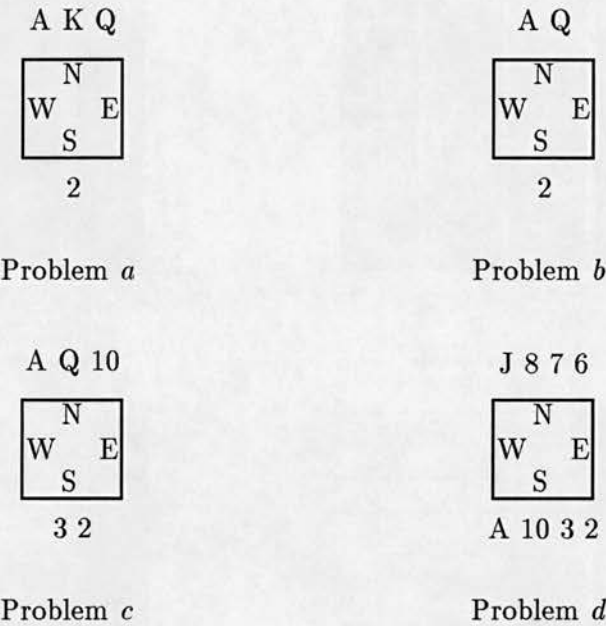


Figure 5.22: Four test problems for single-suit planning

In the first of these problems, FINESSE recognises that the only tactic worth considering is the cashing of a top card, and that the defenders' responses to this can have no effect on the number of tricks won in the suit. There is therefore only one planning state produced on the first planning level of the tree below the root node. Similarly, the second and third levels will contain only single states, so the complete plan will be entirely linear. Carrying a history in this example would therefore not be expected to have any effect (the history would never be checked, as it would always be empty when the single state is examined) and indeed this is the story told (within the bounds of experimental error) by the first bar graph of Figure 5.23. The remaining examples, however, give rise to larger trees, with problem *b* having two initially applicable tactics and a depth of two planning levels below the original root, problem *c* having three applicable tactics and a depth of three, and problem *d* having six applicable tactics and a depth of four.

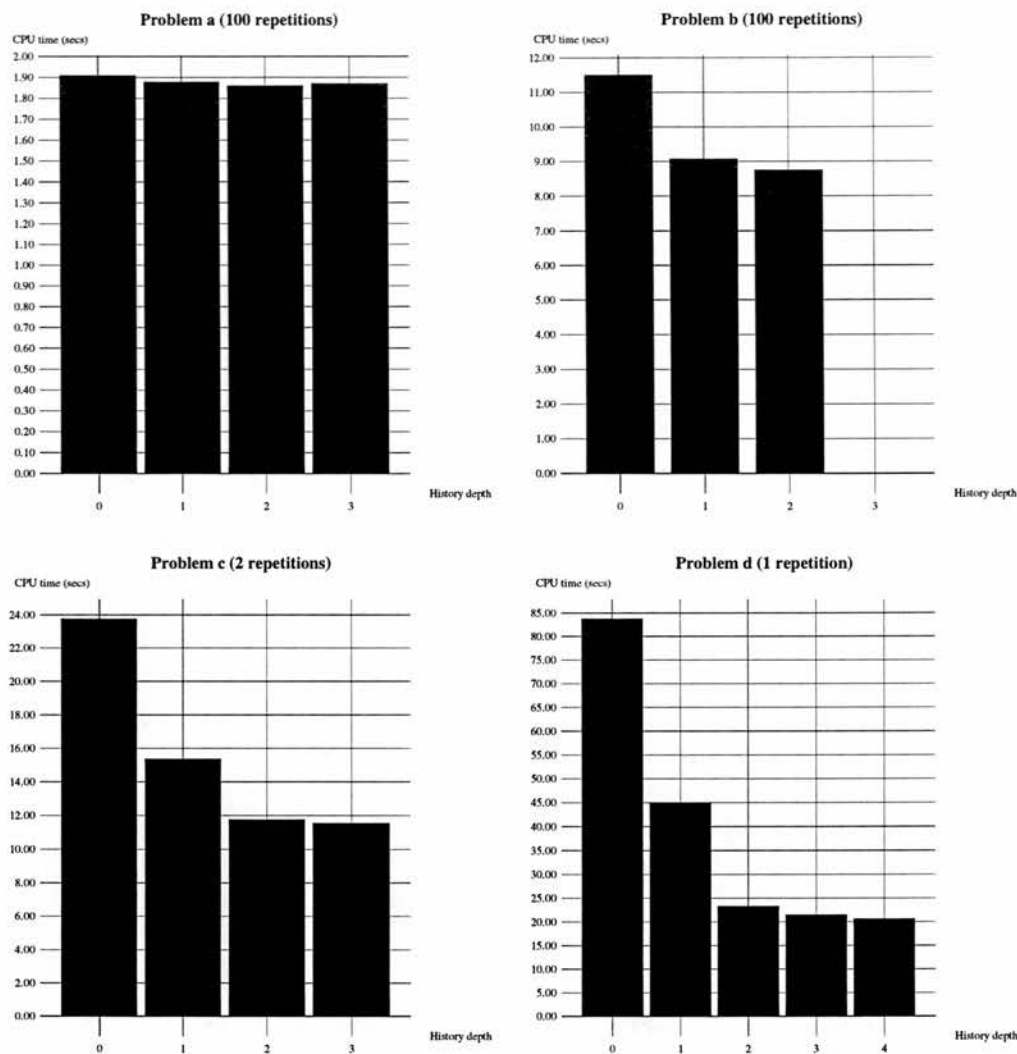


Figure 5.23: Graphs of cpu time against history depth for four single-suit problems

In the light of the analysis of §5.7.1, the results depicted in Figure 5.23 are somewhat surprising. It appears that performance improves as the history list mechanism is extended to lower levels of the tree, with the improvement being marked at first, becoming less significant at levels below half the depth of the search tree, and minimal at the final level. There are two possible reasons why searching through the history lists should fail to become more expensive than checking the method set. One is that our estimate of α_i may have been incorrect. After all, this value has to be small for the analysis of §5.7.1 to hold, and at the lower levels of the tree, where there are a large number of nodes and only a relatively small number of cards remaining with which

to create distinct states, it may well be that the value of $\frac{1}{30}$ is a significant underestimate. A second possibility is that the process of building trees by using pointers to repeated substructures rather than creating fresh copies results in a significant saving of system memory. If the increased efficiency in memory handling can compensate for the expected cost of extending the history mechanism to the lower levels of the trees, a performance benefit may well be the result.

5.7.3 Memory Management

As indicated by the final graph of Figure 5.23, planning with history lists enabled FINESSE to produce plans for even problem *d* in around 20 seconds. This was one of the hardest problems we have encountered, and one of those which originally had often resulted in system crashes. The difficulties encountered with this problem were directly responsible for our discovery of the phenomenon of *non-locality*, which we discuss in the following chapter. By writing the output of the system to a file we were able to measure the size of the plan for this problem, which we found to be 4.3 Mb! To a large extent, the size of this data structure can be put down to the inclusion of substantial amounts of book-keeping information at each node of the tree, and especially to the data which is generated by the algorithm for identifying the optimal choices at nodes where the declarer must select the next action. We will discuss this algorithm in detail in the following chapters. For now, we will simply point out that with data structures of this size, the benefits of not explicitly constructing repeated substructures can be highly significant.

If the data structure is formed utilising pointers, however, it also becomes important that this form is retained. Unfortunately, the initial implementation of the graphical interface ignored this constraint, creating a form of ‘global variable’ by recording the plan structure in the Prolog database. In SICStus Prolog, this action created an actual copy of the *entire* data structure, without respecting those portions constructed using pointers, leading to the sudden consumption of 4.3 Mb of memory. To deal with this problem, the entire system, including the display predicates, was modified to carry pointers to all the currently active plans, without requiring the use of the Prolog database. This allowed inspection of the data structures without the creation

of complete copies. The drawing options available for displaying a tree were also modified to allow just a small portion of large trees to be rendered, and navigation through the tree structure was made possible via the ability to select different root nodes.

This issue may appear to be solely implementational, but the problems involved in readily understanding system output measured on the scale of megabytes are non-trivial. If the tree structure is recorded in the Prolog database (as in CLAM, for example) with records for each node, care needs to be taken to ensure that no nodes are duplicated, identifying any newly produced state that has been examined once before with the previously expanded node. However, such a method of storage would make any alteration of the plan structure by subsequent algorithms an awkward process. Since FINESSE is a modular system, with the output of one stage being passed to the next for further processing (*e.g.*, for the identification of the most promising branch selections), the new implementation therefore refrains from using the Prolog database for this purpose.

5.8 Summary

In this chapter we have introduced the proof-planning paradigm, and in particular the notion of using tactics and methods to restrict the available options at any stage of a state-space planning algorithm. We showed how these ideas could be applied to single-suit play in the game of Bridge, introducing the original version of the FINESSE system of which this thesis is an extension. We presented the seven single-suit tactics used by FINESSE and noted that this set was sufficient to represent the optimal lines of play for all the single-suit situations we have so far encountered. We then detailed how the methods describing these tactics were incorporated into a new planning algorithm capable of constructing the entire tree of legal tactic sequences for any position. We also looked at some enhancements to the representation, the efficiency, and the interface of the system which have been made during the course of this research, producing data and analysis demonstrating a significant improvement in system performance.

Chapter 6

Search in Games with Incomplete Information

The player on the other side is hidden from us. We know that his play is always fair, just, and patient. But also we know, to our cost, that he never overlooks a mistake, or makes the smallest allowance for ignorance.

— THOMAS HENRY HUXLEY
Lay Sermons, *etc*, iii. A Liberal Education

We demand rigidly defined areas of doubt and uncertainty!

— VROOMFONDEL
The Hitchhiker's Guide to the Galaxy

In the previous chapter we described how the search space in single-suit Bridge play problems can be reduced by a planner utilising tactics and methods. In order to actually use the output of such a planner to play the game of Bridge, it is necessary to identify the *best* strategy from amongst the possibilities produced by the planner. The original version of FINESSE [Frank 91] contained no mechanism for this, simply selecting the left-most branch at each choice point (and allowing user-directed backtracking through all possible combinations of selections).

In this chapter we address this issue by presenting a general examination of search algorithms in games with incomplete information. In particular, we identify two problems that can occur in such domains. The framework established here will then be used in the following chapter to suggest both domain-independent and Bridge-specific algorithms for coping with these difficulties.

6.1 Introduction

We base our analysis on the introduction of incomplete information into the standard framework of strategy optimisation for zero-sum two-player games [Nash 51]. To aid analysis of actual games, we formalise a particular *best defence* model of a game which corresponds to the often-made assumption of playing against a rational, informed opponent. We show for such games that an *equilibrium point* for the two players' strategies is well-defined, and we describe an algorithm which we call *exhaustive strategy minimisation* that identifies such equilibrium points. This algorithm formalises the idea of selecting a strategy directly, by explicitly enumerating the possibilities, computing the minimum payoff of each, and then selecting the strategy with the best payoff. By explicitly manipulating strategies, our algorithm has different properties from standard minimaxing and produces more accurate results than adaptations of minimaxing (discussed below) to domains with incomplete information.

Unfortunately, the complexity of exhaustive strategy minimisation makes it impractical on all but the smallest of game trees. In the following chapter, we therefore examine some techniques that may improve performance in some games. Here, though, we use our formalisation to examine approaches like the double-dummy algorithms of Levy and Ginsberg which we introduced in §2.2.4. Recall that the idea of such an algorithm was to tackle the incomplete information in Bridge by examining some significant subset of the possible distributions of cards. In more general incomplete information games, this will be equivalent to examining some subset of the possible *worlds* in which the play make take place. We use our formalisation to show that, even in very simple incomplete information games, this approach can yield suboptimal results against best defence. In particular, we identify two serious problems which afflict such algorithms, independently of how many worlds are considered.

The first of these problems, which we name *strategy fusion*, affects any algorithm which attempts to combine together strategies for particular worlds to produce an optimal strategy across all (or some statistically significant subset) of worlds. The flaw in this approach occurs because of the property of incomplete information games that the exact state of the world at any given point of the play may not be known to a player.

This fact imposes a constraint on a player's strategy that he must behave the *same* way in *all* possible worlds at such points; a constraint which is typically broken when combining together strategies designed for individual worlds.

The second problem, which we name *non-locality*, arises when the players of a game have differing information on the possible worlds. An opponent with knowledge of the likely possible worlds can use this to his advantage. In particular, he can direct play towards the portions of the tree of possible moves which are most favourable in the worlds he expects. Thus, some positions in the game may never be reached under particular worlds (as the opponent may always find better alternatives). In general, determining which nodes in the search space will be reached under which worlds requires examining the entire tree of possibilities (since each move an opponent makes gives him the chance to select different portions of the tree in different worlds). Tree search algorithms, however, are generally 'compositional', in the sense that they determine the best play at an internal node of a search space by analysing only the subtree of that node. Such algorithms (*e.g.*, minimaxing) will not take into account the possibility that under some worlds the play may never actually reach that node. As in strategy fusion, the problem here is one of handling the notion of strategy incorrectly. By just evaluating subtrees, such algorithms consider only *partial* strategies; the complete strategies for the entire game would also have to specify what actions would have been taken in all other nodes outside the subtree.

We show that in incomplete information games, both strategy fusion and non-locality can lead to sub-optimal performance, illustrating our arguments using simple game trees and with actual play situations from Bridge itself. In the context of Bridge, we also show that our model corresponds closely to the form of the game typically analysed in expert texts, thus raising serious questions about the possibility of achieving true expert-level Bridge with the type of architecture suggested by Levy and Ginsberg.

We proceed as follows. In §6.2 we introduce preliminary concepts from game theory and apply these in §6.3 to games with incomplete information; in particular we show how Bridge can be analysed within this framework by introducing a best defence model of such games. We follow this in §6.4 by giving an algorithm for computing optimal strategies in this best defence model. The second half of the chapter then considers

Bridge in some detail: in §6.5 we present proposed architectures for Bridge card play based on the minimax algorithm, and in §6.6 and §6.7 we use our game theoretic framework to identify why such architectures must yield suboptimal results against best defence. Afterwards, we summarise our conclusions.

6.2 Game Theory Background

In this section we introduce definitions and terminology necessary to make the discussion self-contained. This is based largely on the work of [von Neumann & Morgenstern 44] and [Luce & Raiffa 57].

6.2.1 The Extensive and the Normal Forms of 2-player Games

In its *extensive* form, a game is a finite tree in which each node corresponds to a *move* where a selection between the branches is made. Each node is identified as being either a *personal* move or a *chance* move. Personal moves are made freely by one of the players, creatively named ‘1’ and ‘2’, since we will consider only two-player games. Chance moves are decided by some mechanical device (*e.g.*, the shuffling of a pack of cards, or the tossing of a coin) that selects a branch in accordance with definite probabilities. There is one distinguished node which represents the start of the game: a *play*, α , of the game involves starting at this node and allowing each of the players (or chance) to choose a branch until a leaf node of the tree is reached. The value that each player i attaches to the outcome of a play α , (*i.e.*, a leaf of the tree), is given by a numerical utility function $K_i(\alpha)$. This value is sometimes also called the *payoff* and K_i a *payoff function*.

One complication is that at any particular move a player may not have full knowledge of the choices made prior to that point in the play. For example, in many card games the play begins with the shuffling and dealing of a pack of cards into hands, which are not visible to all players. Also, the outcome of personal moves may be hidden from other player(s), such as when a card is played face-down. At any move, then, it is possible that a player will be unsure of the actual position of the play within the game tree. To precisely formalise the extensive form of a game, therefore, requires the

nodes of the tree to be partitioned into sets of nodes between which a player will not be able to distinguish. We will follow [Luce & Raiffa 57] in referring to these sets as *information sets*. We will also model our actual definition on that of [Luce & Raiffa 57, pages 39–51], requiring the specification of a 2-player game in extensive form to include the following:

- A finite tree, t , with a distinguished node (the first move in the game).
- A partition, $\mathcal{P}(n)$, of the nodes, n , of the tree into three sets. These sets tell which of the two players (1 or 2) or chance (0) selects the next move at each node.
- A probability distribution over the branches of each chance move, defined by assigning a probability $\pi(n)$ to each daughter of a chance node.
- A refinement of the player partitions into information sets, $\mathcal{I}_i(n)$, for each player i . Each node, n , at which $\mathcal{P}(n) = i$ is classified by $\mathcal{I}_i(n)$ into one of the sets of nodes (numbered as integers $1, 2, \dots$) between which player i will not be able to distinguish.
- An identification of corresponding branches for each of the moves in each of the information sets. (Since a player cannot distinguish between nodes in an information set, the possible moves will appear the same to him at each node of a set. When constructing the tree representation of the extensive form, we must therefore indicate which branch at each node of an information set corresponds to the ‘first’ possible move, the ‘second’ possible move, and so on. In our diagrams, we will assume that this identification is in simple left-right order.)
- For each player, i , a numerical utility function, K_i , defined over the set of end points of the game tree.

Figure 6.1 gives an example of the extensive form of a two-player game that starts with a chance move (represented by a diamond). This move has five possible outcomes, and is followed by a personal move of one of the players (represented by a circle). The information set for this player, whatever the outcome of the first chance move, contains

only one node, *i.e.*, there is no ambiguity over his actual position in the game tree. This means that the outcome of the initial chance is known to him. For the second player's moves (represented by squares), however, there are two information sets. This is because he is only aware of the outcome of the previous player's move, and not of the outcome of the initial chance move. The payoff he will receive (represented by the numbers at the leaf nodes) therefore depends on information which is not known to him.

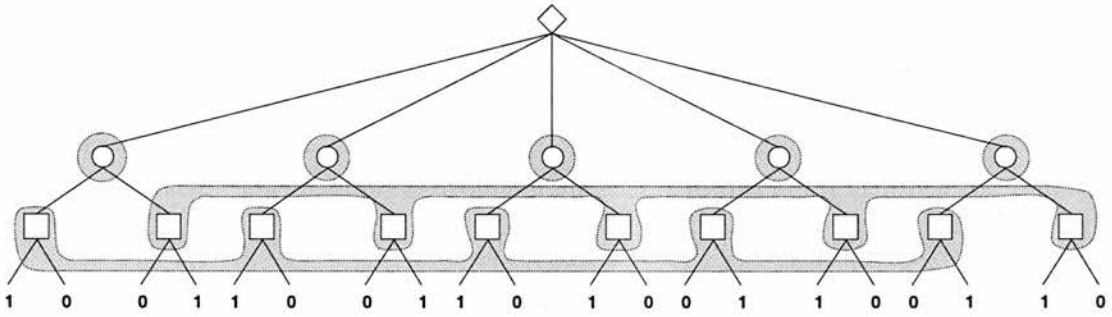


Figure 6.1: Information sets in the extensive form of a two player game with one chance move

Most games will be represented by a game tree that is too large to enable the extensive form to be given in practice. In order to facilitate mathematical analysis it is therefore common to work instead with an equivalent formalisation called the *normal form*. This formalisation forces each player, before the game starts, to state in advance what choices they would make in *any situation that could possibly arise during the course of the game*. Such a specification forms a *strategy*. Given the extensive form of a game, an easy way to formulate the possible strategies for each player is to assign a number $1, \dots, r$, to each branch stemming from a node with r branches. A strategy for a player with q information sets can then be represented by a q -tuple in which each element corresponds to the move to be made in one of the sets. Utilising this notion of strategy, a 2-person game in normal form is defined by specifying:

- two *strategy spaces* X_1 and X_2 , which are the respective sets from which the two players can choose their strategy, and
- two real-valued payoff functions $K_1(x_1, x_2)$ and $K_2(x_1, x_2)$ which give the utility

for each player when strategy x_i is selected by player i . If there are no chance moves in the game, then a given strategy selection (\hat{x}_1, \hat{x}_2) determines a unique play of the game, α , and we can define $K_i(\hat{x}_1, \hat{x}_2) = K_i(\alpha)$. If the game contains chance moves then (\hat{x}_1, \hat{x}_2) instead impose a probability distribution over all the possible plays. If we use $prob(\alpha)$ to denote the probability of play α occurring when (\hat{x}_1, \hat{x}_2) are chosen, the payoff for each player is expressed in terms of the mathematical expectation $K_i(\hat{x}_1, \hat{x}_2) = \sum_{\alpha} prob(\alpha)K_i(\alpha)$.

Formulated in this way, the course of a single play of a game first involves the choice of a strategy by each player. In *non-cooperative* games, this choice is made without any pre-play communication between the players. The basic theory of non-cooperative games is based of the concept of an *equilibrium point*, due to J. Nash [Nash 51]. This views any selection of strategies by each player as being a ‘solution’ to a game whenever no single player can individually increase his payoff, or expected payoff, by changing his strategy selection. That is, a pair (\hat{x}_1, \hat{x}_2) is an equilibrium point if the following hold.

$$\begin{aligned} K_1(\hat{x}_1, \hat{x}_2) &\geq K_1(x_1, \hat{x}_2) \quad \forall x_1 \in X_1, \\ K_2(\hat{x}_1, \hat{x}_2) &\geq K_2(\hat{x}_1, x_2) \quad \forall x_2 \in X_2. \end{aligned} \tag{6.1}$$

A special case, relevant to our domain, is that of *zero-sum games*, where $K_1(x_1, x_2) + K_2(x_1, x_2) = 0$, for all $x_1 \in X_1, x_2 \in X_2$. Under this condition, we can arbitrarily select a K_i (we choose K_1) and rewrite (6.1) as

$$K_1(x_1, \hat{x}_2) \leq K_1(\hat{x}_1, \hat{x}_2) \leq K_1(\hat{x}_1, x_2), \forall x_1 \in X_1, x_2 \in X_2. \tag{6.2}$$

This states that K_1 must have a saddle point; we consider next when this holds.

6.2.2 Minorant and Majorant Games: The Minimax Theorem

Let Γ represent a non-cooperative, zero-sum, two-player game where the players pick their strategies without knowledge of that of their opponent. To analyse such games, von Neumann and Morgenstern introduce two variations on Γ . The first of these, Γ_1 ,

is defined so that it agrees with Γ in every detail except that player 1 must choose x_1 *before* 2 chooses x_2 , so that 2 makes his choice in full knowledge of the particular x_1 decided on by 1. Since 1 is at a disadvantage in this game compared to his position in the original game, Γ_1 is termed the *minorant* game of Γ . The second variation, Γ_2 , is the dual whereby 2 chooses his strategy first; this game is termed the *majorant* game of Γ .

For these new games, Γ_1 and Γ_2 , the identification of the optimal strategy is simplified, as the 'best way of playing' may be given a clear meaning. Let us consider the minorant game Γ_1 , in which 1 is the first to select a strategy. For any particular selection, \hat{x}_1 , 2 will choose an x_2 to minimise the value of $K_1(\hat{x}_1, x_2)$. Thus, when 1 is choosing an x_1 he can be sure (assuming a competent opponent) that the expected outcome of the game is $\min_{x_2} K_1(x_1, x_2)$. This formula is a function of x_1 alone, and since 1 is attempting to maximise his payoff, the best expected outcome he will therefore be able to achieve is

$$v_1 = \max_{x_1} \min_{x_2} K_1(x_1, x_2).$$

A similar argument shows that if both players play the majorant game well, the resulting payoff that can be expected by 1 is

$$v_2 = \min_{x_2} \max_{x_1} K_1(x_1, x_2).$$

Von Neumann and Morgenstern show that these values can be used to establish upper and lower bounds on the value, v , that 1 can hope for from a play of Γ itself. This result is then refined to produce a theorem that states that there is a subclass of zero-sum two-player games (those with perfect information) for which $v_1 = v = v_2$. This is equivalent to writing

$$\max_{x_1} \min_{x_2} K_1(x_1, x_2) = \min_{x_2} \max_{x_1} K_1(x_1, x_2), \quad (6.3)$$

which is in turn logically equivalent to (6.2) (*i.e.*, it states that K_1 has a saddle point). The form of (6.3) has led Von Neumann and Morgenstern's result to be referred to

as the *minimax theorem* and it forms the basis of the well-known minimax algorithm suggested by Shannon [Shannon 50].

6.2.3 Pure and Mixed Strategies

The minimax theorem does not hold for all two-player games. To achieve an existence theorem for general games, Von Neumann and Morgenstern extend the notion of strategy as follows. If the sizes of the sets X_1 and X_2 are m and n respectively, the players, rather than choosing an x_1 and an x_2 from these sets, instead specify vectors $\vec{p} = (p_1, \dots, p_m)$ and $\vec{q} = (q_1, \dots, q_n)$ ($p_i, q_i \geq 0, \sum p_i = 1, \sum q_i = 1$) where p_i gives the *probability* that 1 will select the i th member of X_1 as his strategy and q_i is the probability that 2 will select the i th member of X_2 . When the players select their strategies in this probabilistic manner, the natural interpretation of the expected outcome is the mathematical expectation

$$K(\vec{p}, \vec{q}) = \sum_{i=1}^m \sum_{j=1}^n K_1(x_i, x_j) p_i q_j .$$

As in the previous section, it is possible to show that for this augmented game, the function K has a saddle point. This is a probability theoretic interpretation of the previous saddle point theorem, and illustrates that in some games there is a definite disadvantage to having your strategy ‘found out’ by the opponent. Using a probability vector to select randomly from among a number of possible strategies affords protection from exactly such an occurrence. Strategies in this augmented sense are called *mixed strategies*. The strategies of the previous section are a special case of mixed strategies in which the probability distribution is a 1-point distribution, and are referred to as *pure strategies*.

6.2.4 Preliminarity and Anteriority

Let us consider again the extensive form of a game. If we represent a particular play of a game as a sequence of moves M_1, M_2, M_3, \dots , we can define the moves which are *anterior* to some personal move M_k as being any move M_i with $i < k$. Notice that this property is transitive, *i.e.*, if M_μ is anterior to M_λ and M_λ is anterior to M_κ , then

M_μ is anterior to M_κ . We can also look at the amount of information on the outcome of the anterior moves that is available to the player who is called upon to make move M_k . It is possible that this player will know which branch was chosen for each of the moves M_1, \dots, M_{k-1} , but it may also be that he has only partial knowledge, or no knowledge at all. The simplest way to describe a player's state of information at move M_k is to form a set of *preliminary* moves, \mathcal{P} . This set consists of the moves M_i , for some $i \in \{1, \dots, k-1\}$, such that the branch chosen for any of the $M_i \in \mathcal{P}$ is known to the player, but the exact choice made at any of the other anterior moves is not.

The class of games in which preliminaryity and anteriority coincide (*i.e.*, where a player called upon to make a move is informed about the outcome of all the anterior moves) is called *perfect* information games. We have already seen in §6.2.2 that the minimax theorem enables each player's optimal strategy to be interpreted in a precise way for such games. However, in games where anteriority does not imply preliminaryity (which we call *incomplete information* games), peculiar features can result. For instance, the property of preliminaryity need not be transitive, as illustrated by the following example, which we quote from [von Neumann & Morgenstern 44, page 52]:

Poker: Let M_μ be the deal of his 'hand' to player 1—a chance move; M_λ the first bid of player 1—a personal move of 1; M_κ the first (subsequent) bid of player 2—a personal move of 2. Then M_μ is preliminary to M_λ and M_λ to M_κ but M_μ is not preliminary to M_κ (*i.e.*, 1 makes his first bid knowing his own 'hand'; 2 makes his first bid knowing 1's (preceding) first bid; but at the same time 2 is ignorant of 1's 'hand'.)

This intransitivity of preliminaryity involves both players, but it is also possible that preliminaryity could be intransitive among the personal moves of one particular player. Bridge provides an example of this, since although it is played by four players, the rules of the game dictate that these players form two teams, which play against each other. Again using a description from [von Neumann & Morgenstern 44, page 53]:

Bridge is a two-person game, but the two players 1 and 2 do not play it themselves. 1 acts through two representatives A and C and 2 through two representatives B and D . Consider now the representatives of 1, A and C . The rules of the game restrict communication, *i.e.*, the exchange of information, between them. *E.g.*: let M_μ be the deal of his 'hand' to A — a chance move; M_λ the first card played by A — a personal move of 1; M_κ the card played [...] by C — a personal move of 1. Then M_μ is preliminary

to M_λ and M_λ to M_κ but M_μ is not preliminary to M_κ . Thus we again have intransitivity, but this time it involves only one player.

Intransitivity of preliminaryity raises the possibility of signalling (*i.e.*, the spreading of information to other players). In Bridge, players who form one team but cannot see each other's cards will wish to promote this signalling, and an elaborate system of conventional signals has been developed to enable this. In Poker, the interest of a player lies in preventing this signalling, and this is usually achieved by irregular and seemingly illogical behaviour when making a choice. The first of these two types of procedures is *direct* signalling and the second is *inverted* signalling.

6.3 Equilibrium Points in Bridge

With the definitions of the previous section behind us, we can now formalise the particular problem that we address in this chapter. Since our motivation is the study of Bridge as a game with incomplete information, we first describe the game's characteristics in more detail, and then present some simplifying assumptions that will aid analysis of the problem.

6.3.1 Bridge as a Game of Incomplete Information

Analysis of the game of Bridge is extraordinarily complicated. The shuffling and dealing of the pack of cards at the start of the game in effect selects one of $52!/13!^4$ possible positions for the subsequent play (the order of the cards in a hand does not matter — hence the dividing factors). Further, each player can initially see only their own hand, so (as we have already seen) preliminaryity will be intransitive among the moves of the players, giving the opportunity for both direct signalling between partners, and inverted signalling to confuse the opponents. A related problem caused by this mis-match between the available information is that of predicting an opponent's beliefs. Korf, for example, although not motivated by considering games with incomplete information, examined the situation where two players have evaluation functions which are not known to the other [Korf 89]. Such a game can be viewed either as a zero-sum game in which the difference in evaluation functions is due to their heuristic nature, or as a non-zero-sum game in which the evaluation functions represent the payoffs that each player

would actually receive, or as an incomplete information game in which the difference in evaluation functions is due to the differing information available to each player. Korf's description of this situation uses the common convention of naming the two players MAX and MIN. In the context of the previous section where we (arbitrarily) chose K_1 as the payoff function, MAX will be player 1, since he tries to maximise the value of K_1 . Similarly, MIN will be player 2, since he tries to minimise the value. The decision process in a three-level MAX-MIN-MAX tree is described by Korf as follows:

MAX's decision will be based on what he thinks MIN will do. However, MIN's decision will be based on what he thinks MAX will do two levels down. Thus, MAX's decision is based on what MAX thinks that MIN thinks that MAX will do. Therefore, the evaluation function that is applied to each of the frontier nodes is MAX's model of MIN's model of MAX's evaluation function, and the nodes with the maximum values are backed up to the MAX nodes directly above the frontier. Next, MAX's model of MIN's evaluation is applied to the backed up nodes, and the nodes with the minimum values are backed up to the MIN nodes directly below the root. Finally, MAX's evaluation is applied to these backed up nodes to determine the final move.

In Bridge, then, we can identify the following four related complications over perfect information games. First, the intransitivity of preliminaryity between the moves of the two sides will lead us to encounter the problem of ever-deepening levels of reasoning about the opposing side's beliefs. Second, as we saw in §6.2.4, this intransitivity also makes inverted signalling possible, making it advantageous for one side to attempt to prevent the spread of information about their position to the other side. Third, there is the opportunity for direct signalling, in which the two players who form the opposition play to increase each other's information of their side's situation. Finally, the absence of perfect information will entail solutions which are mixed strategies, since using a pure strategy will typically give the opponents an advantage if they can 'find out' what this strategy is. The probabilistic nature of mixed strategies prevents the opponents from knowing which pure strategy will be followed, even if they are aware of the exact mixed strategy that will be used.

6.3.2 The Best Defence Model of an Incomplete Information Game

The problems outlined above present serious difficulties when combined with game trees the size of those generated in Bridge (recall our examination of the search space size in Chapter 4). Thus, rather than tackling the actual game directly, we will make the following simplifying assumptions.

- A-I MIN has perfect information (*i.e.*, preliminary and anteriority coincide for MIN). This cuts the cycle of reasoning about beliefs since now MIN knows exactly what information MAX possesses at any stage of the game, and MIN's model of MAX's evaluation function will be MAX's function itself. Further, this assumption removes the need for MAX and MIN to signal information about their holdings.
- A-II MIN chooses his strategy after MAX. This is the same assumption as that made by Von Neumann and Morgenstern in reducing a game to its minorant form. As indicated in §6.2.2, the outcome of the game can then be specified as a function of MAX's strategy only, thus aiding formal analysis.
- A-III The strategy adopted by MAX is a pure strategy (as defined in §6.2.3). This enables us to search for the optimal strategy from among a finite (although possibly very large) set, in contrast to the set of possible mixed strategies.

We will refer to the result of transforming any game by making these modifications as the *best defence* model of the game. It may seem that the assumptions are so severe that analysis of the best defence version will tell us very little about the original game. However, we can justify the transformation by noting that, at least in the context of Bridge, it is very common to see this best defence model of the game implicitly used in expert texts, which often analyse play problems under exactly the assumptions given formally above. For any given example, such texts typically start by selecting a small number of promising strategies. For each of these strategies, its chance of success is then found by enumerating the possible card distributions and analysing the return produced when the opponents are allowed to choose their best strategy in each perfect information situation. The Bridge Encyclopedia [ACBL 94], for example, devotes 56

pages to presenting the best strategies (with percentage chances of success) for different card combinations, all of which are analysed in this way. Notice that such an analysis is precisely the situation modelled by the assumptions A-I and A-II: first MAX picks a strategy, and then a MIN with perfect information about the outcome of all prior chance moves is allowed to select the best response. This kind of MIN opponent is one which knows the way in which MAX plays, and will always use this knowledge to be ready with the best possible response to MAX's actions in any world.

Under the assumptions A-I and A-II, the optimal strategies for MAX will in general be mixed strategies, since this will afford some degree of protection from his actual moves being completely predicted by MIN. However, many Bridge books, including the Bridge Encyclopedia, restrict their consideration to pure strategies as solutions to these situations. This restriction may either be motivated by a desire to make the game easier to analyse (as in our case above), or may simply be an error of oversight. Whatever the reason, such books typically refer to this as choosing the best strategy 'against best opposition' or 'against best defence', and the formulation is presumably popular because the payoff that can be obtained against the strongest opposition can be treated as a kind of lower bound on the payoff which can be expected when the opponents are less accomplished. Since we are not aware of any way of analysing the actual game, with all the inherent difficulties identified in the previous sections, we will follow the tradition of using an analysis of the best defence model to produce useful insights into the actual game itself. However, we should point out that the analysis that follows, which is designed to identify the best *pure* strategies for MAX, could in turn be used to generate good *mixed* strategies. The identification of a small number of good (pure) strategies for each player makes it feasible to form a matrix of the payoffs produced for each possible choice of these strategies by the declarer and the defenders. This matrix can then be solved using established techniques, for example by reducing the vector maximisation problem set up by the matrix to a set of constrained scalar maximisation problems that can be solved by techniques of mathematical programming. The strategies produced in this way would be solutions to the probability theoretic interpretation of the minimax theorem introduced in §6.2.3.

6.3.3 Solving the Best Defence Model

What does it mean to solve the best defence model of an incomplete information game? One of the main differences between the best defence model and the actual game is that MIN chooses his strategy after MAX. This makes the best defence model similar to the minorant form of a perfect information game, for which, as we saw in §6.2.2, the value to be expected by MAX (player 1) is:

$$v_1 = \max_{x_1} \min_{x_2} K_1(x_1, x_2). \quad (6.4)$$

For the minorant game, $\min_{x_2} K_1(x_1, x_2)$ is a function of x_1 , so it is easily maximised. In our best defence model, however, there is a complication. MIN (player 2) now has perfect information. He therefore always knows his current position in the game tree, and for any choice of x_1 by MAX he can select an optimal x_2 .

MAX, however, is not party to the same information as MIN, so he will be unable to directly identify whether any x_2 will be optimal for MIN. Specifically, since he does not have perfect information, there may be moves in the game for which MIN knows the outcome, but MAX does not. Selection of an optimal x_2 will require knowledge of the outcome of these moves. In the following section we will introduce an algorithm which deals with this problem by identifying (possibly different) choices of x_2 which are optimal under *each* of the possible outcomes of these moves. After, we discuss the implications of this for games with incomplete information in general, and also more specifically for some algorithms which have been proposed for the game of Bridge.

6.4 Exhaustive Strategy Minimisation

The algorithm we propose comes from directly computing (6.4) with respect to the best defence model: we enumerate all strategies x_1 for player 1 (MAX) and for each of them we separately and exhaustively evaluate $\min_{x_2} K_1(x_1, x_2)$ by examining each possible x_2 under each possible outcome of the chance moves. Since this carries out a minimisation operation for each MAX strategy, we call the algorithm *exhaustive strategy minimisation*.

6.4.1 The Algorithm

Assume that for t a tree, $node(t)$ returns its root node and $sub(t)$ computes the set of immediate subtrees of $node(t)$. Figure 6.2 then defines exhaustive strategy minimisation. Note that this algorithm is very simply defined to directly compute (6.4). It is therefore not necessarily the best implementation in terms of efficiency — an issue we will discuss in more detail in §6.4.5. The assumptions involved in modifying a game

Algorithm $esm(\Gamma)$: Returns optimal strategies for player 1 (MAX) in the best defence, extensive form, 2-player game Γ

- Form the set of player 1's strategies, S , as q -tuples in which the i th entry represents the branch that is chosen at all nodes, n , for which $I_1(n) = i$.
- For each $s_j \in S$, calculate $E_j = esm(t, s_j)$.
- Return the strategy (or strategies) s_j for which E_j is maximum.

where, $esm(t, s)$ takes the following actions, depending on t .

Condition	Result
t is leaf node	$K_1(t)$
$\mathcal{P}(node(t))$ is 2 (i.e., MIN to move)	$\min_{t_i \in sub(t)} esm(t_i, s)$
$\mathcal{P}(node(t))$ is 1 (i.e., MAX to move)	$esm(t_i, s)$, where i is the $I_1(node(t))$ th element of s .
$\mathcal{P}(node(t))$ is 0 (i.e., chance move)	$\sum_{t_i \in sub(t)} \pi(node(t_i)) esm(t_i, s)$

Figure 6.2: The exhaustive strategy minimisation algorithm

into its best defence form are central to the correctness of this algorithm. Firstly, to be able to actually form the set of MAX strategies, S , assumption A-III restricting consideration to pure strategies is needed. Secondly, in fixing a particular $s_j \in S$ and calling $esm(t, s_j)$, assumption A-II that MIN selects his strategy after MAX is required. Finally, assumption A-I (that MIN has perfect information) is used at chance and MIN nodes, since it is only valid to assume that the evaluation of a particular node depends just on the subtree of that node if there is no ambiguity over the position in the tree.

Under these assumptions, it follows by induction on the height of game trees t , that for any strategy s for MAX, $esm(t, s)$ computes the minimal payoff to which MAX can be restricted by MIN. Therefore, given the top-level maximisation loop, the algorithm returns the optimal strategy for MAX and thus correctly computes (6.4).

6.4.2 An Example

To illustrate this algorithm, we will consider again the example of Figure 6.1. Let us interpret this diagram under the common convention that nodes where it is MAX's turn to move are represented as squares, and those where it is MIN's turn are represented as circles. MAX therefore has two information sets, with two possible moves in each, allowing him four strategies. These strategies can be identified by the tuples (1,1), (1,2), (2,1), and (2,2). Let us assume that the 'lower' of the information sets in Figure 6.1 is the set '1', and the 'upper' is the set '2': the first of the tuples above then corresponds to selecting the left-hand branch at every MAX node; the second corresponds to choosing the left-hand branch at all the MAX nodes in the lower set and the right-hand branch in the 'upper' set, and so on.

In (6.4), the result of each of MAX's possible strategies is found by minimising over all the possible responses by MIN. The $esm(\Gamma)$ algorithm models this by calculating $esm(t, s)$ for each possible MAX strategy, s . Let us consider how this will function for the strategy (1,1). The original call to the $esm(t, s)$ algorithm will examine the root of the tree, find it to be a chance node, and make recursive calls on each of the subtrees. Each of these subtrees has a MIN node at the root, so further recursive calls will then be made on each of their subtrees. The roots of these trees are now MAX nodes, so the branch to select is recovered from the strategy under consideration, and found to be branch 1. Further recursive calls on the subtrees along these branches encounter leaf nodes, at which point the payoffs are returned. These payoffs are then passed back up the tree. At the MIN nodes, the minimum of the subtree evaluations is returned. This process is depicted in Figure 6.3, which shows that the strategy (1,1) will lead to a payoff of 0 under each outcome of the chance move except the third. If the outcomes of this move are all equally likely, then, the evaluation of this strategy is $1/5$.

Examining the remaining strategies in the same way shows that the strategy (1,2) leads

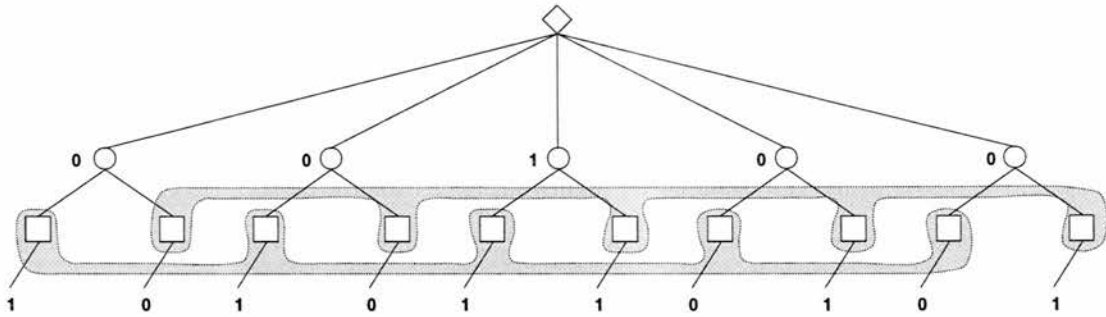


Figure 6.3: MAX's expected payoffs when selecting strategy (1,1)

to a payoff of 1 in just the first two of the outcomes of the chance move, strategy (2,1) leads to a payoff of 1 in just the final two outcomes, and strategy (2,2) always produces a payoff of 0. The two strategies which maximise (6.4) are therefore (1,2) and (2,1), if we assume equally likely outcomes for the chance move.

6.4.3 Comparison with Standard Minimaxing

The exhaustive strategy minimisation algorithm should not be confused with minimaxing, which correctly produces the game-theoretic value of any finite tree in which the players have perfect information. For comparison, in Figure 6.4 we give a formalisation of the minimax algorithm on games in extensive form.

Notice that where the $esm(\Gamma)$ algorithm explicitly manipulates strategies by passing them as arguments and analysing them, the $mm(\Gamma)$ algorithm builds a MAX strategy for each of the possible outcomes of the chance moves by determining a course of action for each MAX node on the basis of the subtree with the largest minimax value. It should be clear that this approach will always produce an evaluation which is greater than or equal to the maximum E_j produced by the $esm(\Gamma)$ algorithm. To see this, consider any MAX node (the only node at which the actions of the $mm(\Gamma)$ and $esm(\Gamma)$ algorithms differ substantially). At these nodes, the $mm(\Gamma)$ algorithm selects the maximum subtree value to back up through the tree, whereas the $esm(t, s_j)$ algorithm must select the branch determined by s_j . The evaluation produced by the $mm(\Gamma)$ algorithm can therefore never be less than that produced by $esm(\Gamma)$. The differences between the two algorithms can be summarised as follows:

Algorithm $mm(\Gamma)$: Take the following actions, depending on t .

Condition	Result
t is leaf node	$K_1(t)$
$\mathcal{P}(\text{node}(t))$ is 2 (i.e., MIN to move)	$\min_{t_i \in \text{sub}(t)} mm(t_i)$
$\mathcal{P}(\text{node}(t))$ is 1 (i.e., MAX to move)	$\max_{t_i \in \text{sub}(t)} mm(t_i)$
$\mathcal{P}(\text{node}(t))$ is 0 (i.e., chance move)	$\sum_{t_i \in \text{sub}(t)} \pi(t_i) mm(t_i)$

Figure 6.4: Simple adaptation of the minimax algorithm to best defence, 2-player, extensive form games

- $mm(\Gamma)$ does *not* respect the constraint imposed by information sets. That is, it does *not* always choose the same branch at nodes which belong to the same information set.
- Since $mm(\Gamma)$ commits to *one* branch selection at each MAX node, it risks incompleteness.

In §6.6 and §6.7 we show that these differences cause difficulties in games with incomplete information. Specifically, we will show that the first leads to the problem of *strategy fusion*, and the second leads to the phenomenon of *non-locality*.

6.4.4 Possible Worlds

The tree of Figure 6.1 contains just one chance node, and this node is located at the root. Let us say that in a tree which contains chance nodes, each possible pure strategy for player 0 (chance) defines a *world state*, or more simply a *world*, in which the play takes place. Consider the special case of a tree which contains just one chance node, and where that chance node occurs at the root of the tree. If, with the exception of the payoffs and the MIN information sets, the subtrees in each possible world are identical, we will say that the tree can be ‘flattened’ by representing the possible worlds in the vertical dimension as differing payoffs at the leaf nodes, rather than in the horizontal

dimension using information sets. For example, Figure 6.5 shows how the tree of Figure 6.1 can be represented if we refer to the possible outcomes (left to right) of the initial chance move as the worlds w_1 , w_2 , w_3 , w_4 , and w_5 .

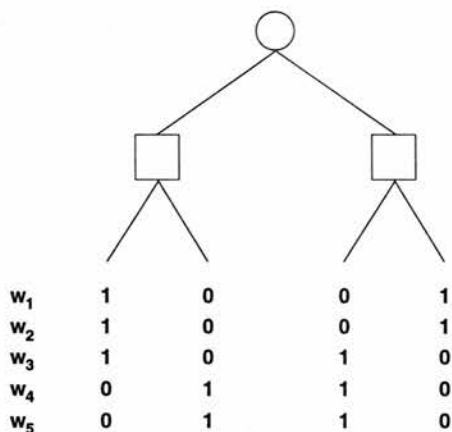


Figure 6.5: Flattened tree of MIN and MAX moves in a domain with five worlds

We will use such trees in the remainder of this thesis to compactly and simply present game trees. Notice that the MIN nodes are the result of identifying together nodes with *distinct* information sets, and that a MIN strategy must therefore allow *different* branch choices under each world. A MAX strategy, on the other hand, must specify the branches to be chosen at each MAX node under *all* worlds.

Note that game trees in Bridge cannot in fact be represented in this way, even though they will always have just one chance move at the top of the tree. This chance move will determine a world, but the possible plays under each of these worlds will not in general be the same. For example, MAX will only be able to play a given card, such as the A_{\spadesuit} , in worlds where he was actually dealt the card to begin with. Thus, the subtrees in each world are different and cannot be ‘flattened’ as described above. The applicability of the $esm(\Gamma)$ algorithm, of course, is unaffected.

6.4.5 The Complexity of Exhaustive Strategy Minimisation

Exhaustive strategy minimisation, while identifying optimal strategies, is not cheap: its complexity is significantly worse than conventional minimaxing. To investigate this, we

first examine how the set of possible MAX strategies, \mathcal{S} , may be generated. We do this by looking at the type of flattened trees we introduced in the previous section, using the complete binary tree of Figure 6.6 as an example. Here, we have numbered the MAX nodes in the order in which they would be encountered during a pre-order traversal of the tree. We will use this numbering to construct 5-tuples which correspond to the possible strategies in the game by virtue of the i th element representing the choice to be made at the node numbered i (effectively, the i th information set).

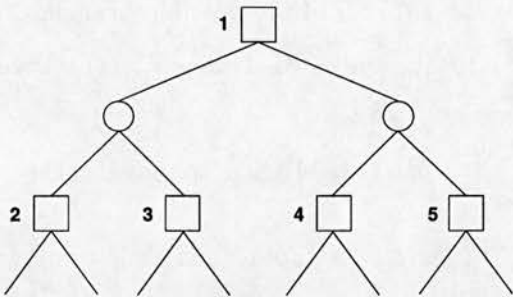


Figure 6.6: Complete binary tree of MIN and MAX moves

To examine how this set of 5-tuples may be generated, let us consider MAX’s possible branch selections. In the strategies where MAX initially selects the left-hand branch at the root of the tree, play is directed to the left-hand MIN node. Since we do not know which branch will be selected at this node, we must now specify branch selections throughout *both* its subtrees. Continuing to examine the left-most branch first, we encounter MAX node 2 and then MAX node 3, where we will assume that we initially again select the left-hand branch. This detail has been glossed over until now, but it should be clear that by selecting the left-hand branches at these two nodes, we will in fact complete a strategy for the game; once the left-hand branch has been selected at the root of the tree, it is no longer necessary to specify branch selections for nodes 4 and 5. We will indicate such superfluous branch selections by an underscore. The strategy we have generated by always selecting the left-most branch, then, is $(1,1,1,_,-)$; overall, the set of possible strategies is:

Left-hand branch at root	Right-hand branch at root
(1,1,1,→-)	(2,→-,1,1)
(1,2,1,→-)	(2,→-,2,1)
(1,1,2,→-)	(2,→-,1,2)
(1,2,2,→-)	(2,→-,2,2)

In general trees, the number of actual strategies in a game is bounded by the number of possible n -tuples. As we have seen above, when encountering a MAX node, any one of the branches may be selected. However, at nodes which are moves of another player, MAX will have to cater for all of the possible branches that may be chosen. Thus, for a given tree, t , the total number of strategies $g(t)$ is given by the following:

$$g(t) = \begin{cases} \sum_{t_i \in \text{sub}(t)} g(t_i) & \text{if MAX is to move at the root of } t \\ \prod_{t_i \in \text{sub}(t)} g(t_i) & \text{if another player is to move at the root node of } t \\ 1 & \text{if } t \text{ is a leaf node} \end{cases}$$

For complete b -ary trees which alternate between the moves of MAX and MIN this formula can be written as a standard recurrence relation. As the addition of an extra layer of MIN nodes to the leaves of a tree does not alter the number of MAX strategies present, we will write this recurrence as a function of the number of MAX levels, n , in a tree. For a b -ary tree with a MAX node at the root, then, we can write the number of MAX strategies, g_n , as

$$g_n = \begin{cases} b & \text{if } n = 1 \\ b(g_{n-1})^b & \text{if } n \geq 2 \end{cases}$$

which has the solution

$$g_n = b^{(\frac{b^n - 1}{b - 1})}.$$

For the example above where $n = 1$, and $b = 2$ this formula gives 8 strategies, as expected. For trees with a MIN node at the root, we can solve a similar recurrence to produce the formula

$$g_n = b^{(\frac{b(b^n - 1)}{b - 1})}.$$

For b -ary trees in general, then, the number of strategies that must be examined is doubly exponential in the number of MAX levels which the tree contains. Further, all of these strategies are examined in *each* of the possible worlds. Thus, the exhaustive strategy minimisation algorithm will require too much computation to be applied to all but the smallest of game trees. However, note that this is an upper bound to the complexity of this problem. We do not know if there is a more efficient way of finding the optimal strategy; however, as we will show in the following sections, minimaxing, and algorithms based on it, are not a substitute.

6.5 Bridge Architectures Based on Standard Minimizing

In the remainder of this chapter we use the above framework to analyse the game of Bridge and in particular to examine the technique, described in Chapter 2, of simplifying the task of card play by solving instead the easier problem where all the players reveal their cards to each other. Since the perfect knowledge situation created by this act is akin to the opponents placing their cards on the table in the same manner as dummy, this scenario is often described as *double-dummy* Bridge. Below, we present the double-dummy architectures proposed by both Levy and Ginsberg [Levy 89, Ginsberg 95]. After, we use our framework to demonstrate why such approaches produce suboptimal results and to formalise two general problems that can afflict search algorithms in games with incomplete information.

A Million Pound Bridge Program?

The first person to explicitly propose the use of a double-dummy solver as the basis for a program to play Bridge appears to have been Levy [Levy 89]. He was confident that this kind of program could win the prize of 1 million pounds offered by former Bridge World Champion, Zia Mahmood, to the designers of a computer player that could defeat him. Here, we describe Levy's algorithm by considering the general problem of selecting MAX's next move in an arbitrary incomplete information game.

Let us say that, for some move under consideration, the set of worlds which are consistent with the outcomes of the previous (anterior) moves is given by W . Let us also

say that it is possible to choose an n such that when we randomly generate n members, w_1, \dots, w_n of W , we have sufficient computing resources to find the minimax value of the current (complete information) game tree under each of these worlds. If there are m possible moves, M_1, \dots, M_m , to choose between and we use e_{ij} to denote the minimax value of the i th possible move under world w_j , the situation in this world will be as depicted in Figure 6.7.

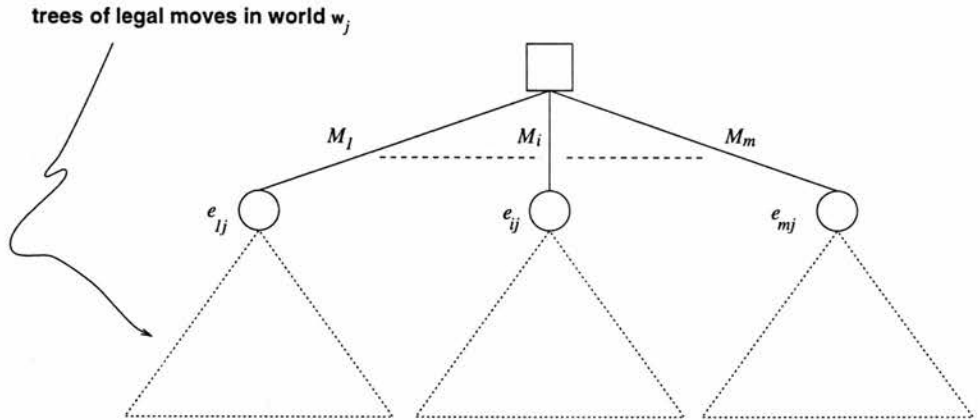


Figure 6.7: The minimax values, e_{ij} , of each move M_i under world w_j

Levy's proposal was that each legal move, M_i , could be given a score based on its expected payoff. In the context of Figure 6.7, Levy's score can be expressed as the scoring function, f :

$$f(M_i) = \sum_{j=1}^n e_{ij} \text{prob}(w_j).$$

Selecting a move is achieved by actually using the minimax algorithm to generate the values of the e_{ij} s, and determining the M_i for which the value of $f(M_i)$ is greatest. Since this technique relies on repeated applications of the minimax algorithm to problems with perfect information, we will refer to it as *repeated minimaxing*, and to the limiting case where *every* possible world is examined as *exhaustive minimaxing*.

Of course, other possible definitions for the scoring function f can be envisaged, and indeed Ginsberg's notion of selecting 'the play that works best on average' [Ginsberg 95] suggests the following alternative:

$$f(M_i) = \sum_{j=1}^n \left(e_{ij} \stackrel{?}{=} \max_k e_{kj} \right),$$

where k ranges from 1 to m , and $\stackrel{?}{=}$ is an infix function which equals 1 if both sides are equal, and 0 otherwise (that is, a move is given a score of 1 for each world in which it is the best, or equal best, alternative). Another possibility (particularly in Bridge, where the objective is usually to win at least a certain number of tricks) is to only consider moves which guarantee a result at least as large as some minimum value, e , say. That is:

$$f(M_i) = \sum_{j=1}^n \left(e_{ij} \stackrel{?}{>} e \right) e_{ij} \text{prob}(\mathbf{w}_j).$$

However, more important than the choice of the scoring function in our current context is the answer to the more fundamental question of whether Levy is justified in speculating that this kind of architecture is the key to playing the cards ‘perfectly’ (Levy’s quotes). Below we show that no possible scoring function can work optimally.

6.6 Repeated Minimizing Fails: Strategy Fusion

The repeated minimaxing architecture described above is based on looking at a representative sample of possible distributions of outstanding cards and using them to evaluate the best strategy. We use the framework we developed in §6.3 to show that this is not possible, *i.e.*, suboptimal strategies may be returned. Indeed, we show that even an exhaustive minimaxing algorithm, examining *all* the possible distributions, may fail to select the correct strategy! Although our discussion is driven by the problems experienced by repeated minimaxing, we also show that any algorithm which shares specific characteristics with minimaxing will experience the same difficulties. We use examples from Bridge to show how such sub-optimal algorithms lead to improper play in real games.

Let us return to the flattened version of the game tree we first introduced in §6.2, this time labelling the nodes and adding one extra possible path, as shown by the nodes d , e , and f in Figure 6.8. If MAX picks the left-hand branch at node d , he will

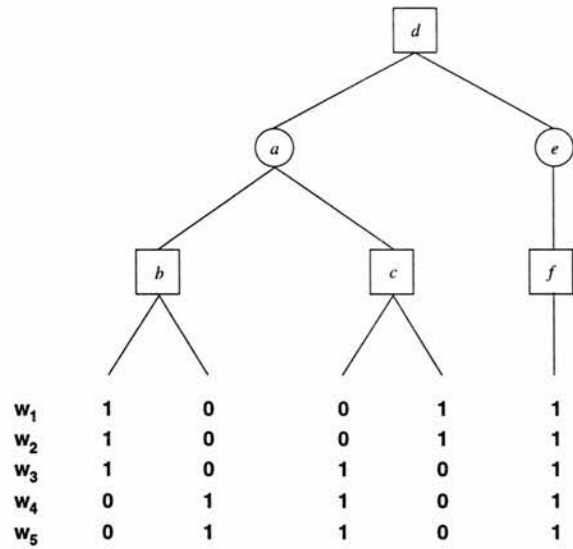


Figure 6.8: A tree of MIN and MAX choices which is mis-analysed by exhaustive minimaxing

encounter the tree with four possible strategies that we have already examined in §6.4. Alternatively, he may select the right-hand branch, which leads to a subtree in which no further MAX choices are necessary. Thus, the extra branch increases the number of possible MAX strategies by one. Since we have already seen that none of the strategies in the left-hand subtree give a payoff of 1 in more than two worlds, MAX's best option at the root of the tree, guaranteeing a return of 1 under every world, is to select the right-hand branch.

However, using the standard minimax algorithm under any single world, the two MIN nodes — a and e — will always have a minimax value of 1 (since in any single world MAX can always choose a branch with a payoff of 1 at nodes b , c and f). Applying repeated minimaxing to the root of the tree then, irrespective of the scoring function used, must assign both the left-hand branch and the right-hand branch the same score. Thus, repeated minimaxing will perform no better than making a random choice between the two moves, even in the limiting case of exhaustive minimaxing. Furthermore, reducing the number of worlds in which the right-hand branch gives a payoff of 1 can produce situations in which repeated minimaxing will usually select the incorrect move, and indeed where exhaustive minimaxing will *always* select incorrectly. For ex-

ample, consider how exhaustive minimaxing behaves in the situation of Figure 6.9(a), where the payoffs on the right-hand branch have been modified so that a payoff of 1 is achieved in only three worlds. Despite this modification, this branch still represents MAX's best move at the root of the tree, under the assumption that all the worlds are equally likely. For exhaustive minimaxing, however, it is the other branch that will be selected by any of the scoring functions of §6.5, as the left-hand MIN node has a minimax value of 1 in all five worlds, whereas the right-hand node has a value of 1 in just three. (Repeated minimaxing *may* pick the correct branch, but only if it does not examine either of the worlds w_1 and w_2 and then makes a fortunate guess). Any *sensible* scoring function will never lead exhaustive minimaxing to select the right-hand branch in this situation, as it clearly cannot be rational behaviour, given a set of alternatives to choose between, to prefer an option whose evaluation is always less than or equal to that of one of the others. To see that no *possible* scoring function can cope with all such situations, consider Figure 6.9(b). To an exhaustive minimaxing algorithm, this tree will be indistinguishable from that of Figure 6.9(a), as the minimax values of the MIN nodes in each are the same under every world. However, in Figure 6.9(a) the best move is the right-hand branch, and in Figure 6.9(b) the best move is the left-hand branch. Any given scoring function will therefore either make the correct choice in just one of these situations or will be unable to distinguish the best move in either case.

The source of the difficulty which repeated minimaxing experiences on these trees lies in a crucial deviation it makes from the exhaustive strategy minimisation algorithm we presented in §6.4. Recall that for each world that it considers, exhaustive strategy minimisation examines the result of *every* possible strategy. Repeated minimaxing, on the other hand, uses the minimax algorithm to find the *best* strategy in a number of worlds. As we pointed out in §6.4.3, using minimaxing in this way does *not* respect the constraint imposed by information sets: it allows *different* strategies to be chosen in *different* worlds. Collecting the minimax values of these strategies and assuming that they represent the payoffs that can be expected under each world ignores the fact that a *choice* of a particular strategy has to be made. We therefore call this problem *strategy fusion*. As we saw in §6.4.3, allowing the minimax algorithm to ignore the constraint imposed by information sets results in evaluations greater than or equal to the correct values (*e.g.*, produced by exhaustive strategy minimisation). Repeated minimaxing

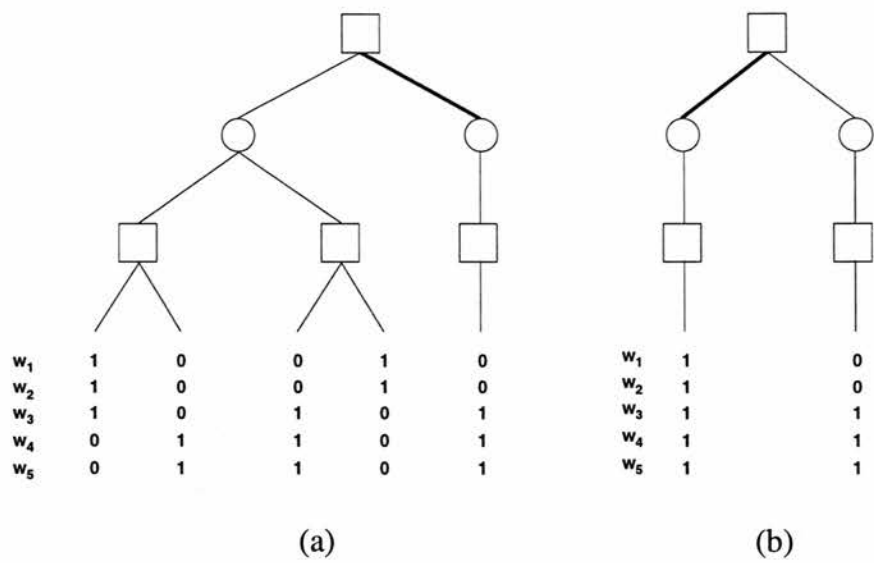


Figure 6.9: Two trees with the best initial move marked in bold

will therefore have the tendency to over-estimate its scoring of any node; the strategy fusion effect leads it to believe that it has the luxury of choosing a different strategy in each world, instead of the best single strategy across all worlds. For example, given perfect information at node b or node c it is easy to determine which branch to select to produce a payoff of 1. Node a will therefore always have a minimax value of 1 under repeated minimaxing. However, we have already seen that when a *single* strategy selection is enforced, the best that can be achieved is a payoff of 1 in at most two worlds.

Another way to visualise the strategy fusion problem is to note that the repeated minimaxing architecture actually models the task of selecting between some number of perfect information *games* each starting with the same chance move. For example, imagine that the subtrees rooted on nodes a and e in Figure 6.8 represent the MIN and MAX moves in a game starting with a chance move that selects one of the possible worlds w_1, \dots, w_5 . Which of these games would we rather play given that they have perfect information (for both players)? This is the question which exhaustive minimaxing answers, and to which repeated minimaxing approximates. It should be clear that such an algorithm will always expect to win the game based on the tree of node a and the game based on the tree of node e . It should also be clear that the

situation modelled by this algorithm is different from the original game as well from the model produced by our assumptions about a best defence model, since it involves assumptions about MAX, as well as MIN, having perfect information.

6.6.1 A Bridge Example

To see an example of strategy fusion in the game of Bridge, consider the situation of Figure 6.10, where we control both the North and the South hands against two opponents who see just their own hand and the North hand, which is the dummy.

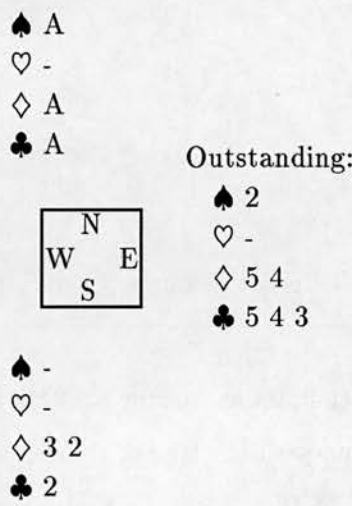


Figure 6.10: A Bridge situation requiring a guess over the best strategy

Assume that spades are trumps — recall that cards in the trump suit beat cards in every other suit — and that it is South’s turn to play. We are worried about the last outstanding trump (the 2♠) but cannot force the opposition to play this card by leading the A♠ because currently it is not North’s turn to play, and South has no cards in the suit. However, whichever opponent has the last trump must also have at least one diamond or one club. Since players must always follow the suit of the card which starts a trick, it is therefore possible to win all the remaining tricks by leading a suit which the opponent with the last trump holds, winning in the North hand with the Ace of that suit, and then clearing the trumps with the A♠. In reality, this choice between leading a diamond or a club is a *guess*, but a double-dummy program will find that it

is *always* possible because it has perfect knowledge of every world. A Bridge situation where repeated minimaxing is misled by strategy fusion can therefore be constructed by adding four cards to the above situation, as in Figure 6.11.

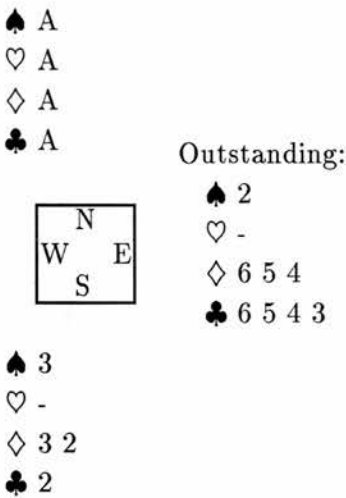


Figure 6.11: A Bridge example where strategy fusion obscures the best strategy

In this new situation, let us assume that the lead is now in the North hand, so that there are four possible moves: the lead of the Ace of any suit. Choosing either of the diamond or the club suits will be (correctly) evaluated by a repeated minimaxing algorithm as less than 100% plays, provided the algorithm examines at least one world in which the Ace may lose to a trump played by an opponent. Choosing the spade suit, on the other hand, will be (correctly) evaluated as a 100% play, since after the A_{\spadesuit} is played there will be no remaining trumps and North's other Aces will be guaranteed winners. However, exhaustive minimaxing will also assess the Ace of *hearts* to be a 100% play. This is because a trump can be played on this card from the South hand (winning the trick) and then the North hand re-entered by making a guess between diamonds and clubs as described above. This play, of course, is clearly not guaranteed to succeed, as would be revealed by an algorithm such as exhaustive strategy minimisation, which would *separately* evaluate the strategies of re-entering the North hand with a club or with a diamond, finding that they would each lose under some of the possible worlds.

6.6.2 Some Practical Consequences

To our knowledge, the concept of strategy fusion has not been formalised before. However, some of the effects of strategy fusion have been discussed in the literature on computer Bridge, although they have typically been mis-analysed. Ginsberg, for instance, presents the card combination of Figure 6.12.



Figure 6.12: An example card combination used by Ginsberg

This is a finessing situation where, if the location of the missing Queen is known, it will always be possible to win four tricks. To see this, consider what happens if we know that West holds the Queen. We start by playing a low card from the South hand, and if West plays the Queen we win with the King and cash the remaining top cards. If West chooses *not* to play the Queen, we win with the Jack or the Ten (a Type 1 finesse), and repeat the same procedure on the following trick. Similarly, we can always win the trick if we know that East holds the Queen, this time by leading low from the North hand. In practice, the location of the Queen is unknown and must be guessed; a wrong guess will allow the Queen to defeat the finesse. However, a double-dummy program will always see the position of the cards and ‘know’ which opponent to play for the card.

Ginsberg correctly realises this problem, pointing out that a double-dummy program will ‘assume that it can always play KJ10x opposite A98x for no losers’. However, he also suggests that such an algorithm ‘won’t mind playing this suit by starting with the 9 (for example)’, whereas human players ‘might play the Ace first to cater to a singleton Queen on our right’. To see why this is not the whole story, consider a game tree composed of just the node *b* in Figure 6.13.

We have already seen that under any single world the minimax value of node *b* is

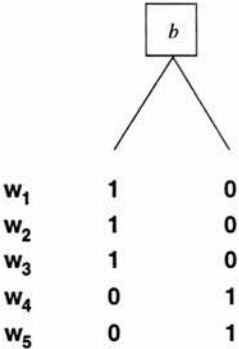


Figure 6.13: A simple tree with just one MAX choice

always 1, and that such nodes can lead to strategy fusion. However, consider this new situation where the node is at the *root* of the tree of possibilities. If we use repeated minimaxing to analyse this tree, the best move is found by determining the branch that has the highest score under some function f . This *separate* examination of each branch removes the freedom to select the best strategy for each world, and since neither of these branches alone can guarantee a payoff of 1 under every world, there *must* now be a payoff of 0 in some worlds. In effect, then, node b represents a point where a *choice* between strategies which win under different worlds has to be made. If such a decision point occurs within a game tree, as in Figure 6.8, a repeated minimaxing algorithm will deceive itself into thinking that it can always make the correct choice. If the decision point occurs at the root of the tree as in Figure 6.13, however, some of the freedom to choose different strategies in different worlds is lost. A repeated minimaxing algorithm will therefore have a tendency to *delay* such decision points wherever possible, as ‘doing it later’ will always appear to be better than ‘doing it now’. Only when it actually reaches a point where a choice *has* to be made will it realise that its previous evaluations were inflated by the effects of strategy fusion.

To see how this is relevant in Ginsberg’s example, observe that in order to be able to always win the finesse against *either* East or West for the outstanding Queen it is necessary to retain both the Ace and the King. Starting a trick by actually playing either of these cards, then, will remove this ability, making it impossible to win all the tricks in the suit under every world. Repeated minimaxing will therefore correctly

evaluate the play of the Ace or the King at less than 100% if it selects any of these worlds to examine. The play of *any other* card in the suit, however, leaves the ability to finesse against either opponent intact, always resulting in an (over-estimated) evaluation of 100%. Therefore, it is not that repeated minimaxing is indifferent over the card to lead in the suit, as Ginsberg suggests, but rather that it will *prefer* to play one of the J, 10, 9, 8, x. Indeed, an exhaustive minimaxing algorithm will *only* choose to play these cards.

Sometimes the delaying of a crucial decision can be important, as the later an actual choice is made, the more information there is likely to be to inform the decision. For example, consider the situation in Figure 6.14, where the aim is to win *all* the tricks and there is no trump suit.

♠ K J 10 x
♥ A K Q
♦ A K Q
♣ x x x



♠ A 9 8 x
♥ x x x
♦ x x x
♣ A K Q

Figure 6.14: An example deal in Bridge. South is declarer, North is dummy and there are no trumps

The spade suit is identical to that of Ginsberg’s example, so there is again a guess over the position of the Queen. However, this is not necessarily a complete guess. As Frank [Frank 89, Page 207] points out in a similar example, ‘by playing out the winners in the other three suits, information will be gained which will increase one’s chance of making the right play, possibly up to certainty’. It might be hoped that the tendency of a repeated minimaxing algorithm to delay such decisions would result in this *discovery play* being made. However, as we have already seen, although the Ace

and King are less than 100% plays, the lower cards in the suit are not, so a repeated minimaxing algorithm may well choose to play the spade suit early.

6.7 Non-locality (Repeated Minimizing Fails Again)

The failure of minimax-like algorithms we address here is more subtle than that of strategy fusion and, to our knowledge, has not been studied before. To illustrate it, we will refer once more to the flattened version of the tree introduced in §6.2, which we repeat here in Figure 6.15 for ease of reference.

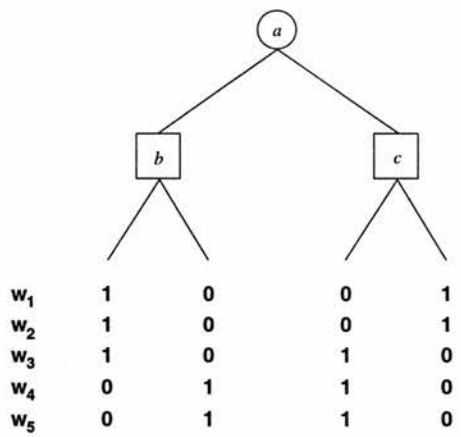


Figure 6.15: Simple tree of MIN and MAX choices in a domain with five possible worlds

When actually playing the game represented by this tree, MAX will only ever have to make one decision: the selection of a move at either node b or node c (effectively, one choice for each of the information sets in the game). Let us again assume equally likely worlds. Irrespective of the MAX node at which the play arrives after MIN's move at the root of the tree, repeated minimaxing, using any of the scoring functions from §6.5, will usually select the left-hand branch (in the limiting case of exhaustive minimaxing, the left-hand branch will always be chosen). Thus, the strategy that will typically be chosen by repeated minimaxing, and indeed the one that will always be selected by exhaustive minimaxing will be the one we have identified as (1,1). However, we have already seen in §6.4 that under the assumption of equally likely worlds, the best strategy selections are actually (1,2) and (2,1)!

The problem here is distinct from that of strategy fusion, and can be traced to a different cause: the way in which a branch selection is made at a node on the basis of an evaluation only of its direct subtree. The inherent assumption in making a branch selection in this way is that the correct move is a function only of the possible continuations of the game. In perfect information situations (*i.e.*, where the position in the game tree is known), this assumption is justified and the minimax algorithm, with its compositional evaluation function, finds optimal strategies for such games. With more than one possible world, however, this assumption is no longer valid. The reason for this is that a decision procedure making this assumption considers only *partial* strategies at any internal node of a tree: the actual strategies themselves would have to specify further choices at other MAX nodes in the tree. This is the manifestation of the incompleteness we alluded to in §6.4.3.

To illustrate the discussion here, let us return to the task of selecting a strategy in our example tree. If we just analyse the subtree of node *b*, we see that the left-hand branch appears to be the best choice because it produces a payoff of 1 in three out of the five possible worlds. In the context of the entire game, however, selecting the left-hand branch at node *b* affects the analysis of node *c*. Since the left-hand branch at node *b* produces payoffs of 0 in worlds w_4 and w_5 , MIN (who chooses his strategy after MAX) will be able to restrict MAX's payoff to 0 in these worlds *irrespective* of MAX's choice at node *c*. Under this circumstance, it is the right-hand branch that is the best choice at node *c*, since it offers a payoff of 1 in two worlds (w_1 and w_2), compared to the single payoff of 1 (in world w_3) offered by the left-hand branch. Similarly, if we consider making a branch selection at node *b* after choosing a branch at node *c*, we find that the best selection is no longer the one which leads to a payoff of 1 in most worlds.

In general, the choice of branch at a given MAX node is not simply a function of the payoffs of the paths which contain the MAX node, but of the payoffs along *any* path in the tree. If MIN can choose a play at a node's ancestor which reduces the payoff (in any world) from that which MAX would expect from examining the MAX node's direct subtree, the best choice of branch at that node may change. For example, in the case of Figure 6.16, the initial selection of branch *a* may be rendered incorrect

if reducing some of the e_{ia} can result in the maximum value of the function f being achieved at a different branch.

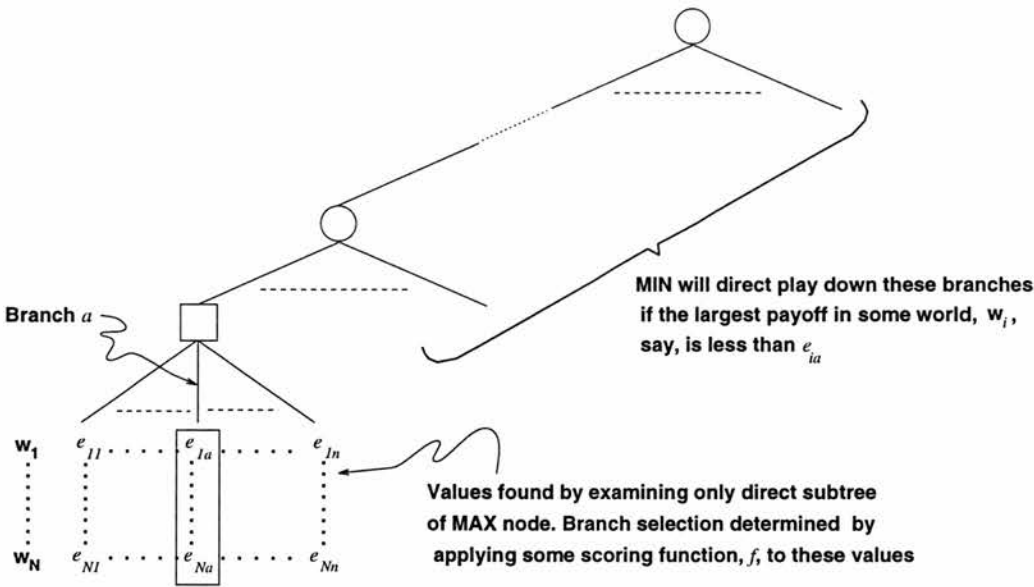


Figure 6.16: A search process in which the selection of branch a may be rendered incorrect by non-locality

We call this problem of having to consider all other nodes in the tree *non-locality*. To see that no possible scoring function can cope with the effects of non-locality, consider the examples of Figure 6.17. The left-hand MAX node is the same in both trees, and leads to a payoff of 1 in just one of the two worlds. However, in one tree the best choice is the left-hand branch and in the other it is the right (since against a MIN player who knows the actual state of the world, the other strategies in each tree always give a payoff of 0). No algorithm analysing these MAX nodes in isolation will be able to make the correct selection in both situations.

Unlike strategy fusion, which is linked with incorrect assumptions about MAX having perfect knowledge, it should be clear that non-locality is closely related to the presence of differing levels of information between the players of the game. MIN's ability to lead the play away from a given node under certain worlds by selecting branches higher in the tree relies both on his ability to choose a strategy after MAX (assumption A-II in our best defence model), and knowledge of the actual state of the world (which under

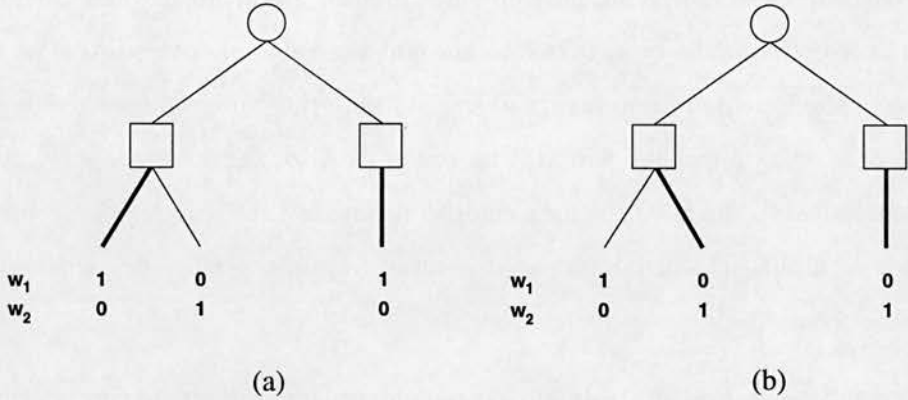


Figure 6.17: Two trees with the best MAX strategy marked in bold

best defence is complete knowledge, from assumption A-I). Under these conditions, it will generally benefit MAX to select mixed strategies rather than pure ones, since this will help prevent his actual moves from being known by MIN. The main limitation of our model is therefore that it restricts consideration to pure strategies. In our case, this restriction was imposed to aid formal analysis, allowing the $esm(\Gamma)$ algorithm to be specified using a simple loop. However, as we noted in §6.3.2, pure strategies are also often given as solutions in standard Bridge texts, and we can further justify this restriction by pointing out that if a number of ‘good’ pure strategies are identified for a game they can then be used to generate mixed strategies which select between them probabilistically.

We should emphasise that since non-locality occurs as a consequence of (the interaction between the assumptions of the best defence model and) the implicit assumption that the best choice at a node can be determined by analysing just the node’s subtree, the phenomenon will be independent of the accuracy of the evaluations on which the branch selection is based. For example, in Figure 6.16, non-locality would still be possible whether the e ’s were produced using repeated minimaxing (risking incorrect values caused by strategy fusion) or by using exhaustive strategy minimisation to find the results of the optimal strategy for the subtree (therefore producing exact values). Thus, the solution to improving the double-dummy architectures of Levy and Ginsberg does not lie with the simple replacement of the minimax algorithm with modified versions, such as average propagation [Nau *et al* 88] or product propagation

[Pearl 81, Pearl 83]. The back-up rules in such algorithms, when calculating a value to propagate up the tree, take into account the value of *each* subtree at a node. Their use in the repeated minimaxing algorithm may therefore reduce the effects of strategy fusion, but non-locality will still be present. Also, the very property that each node contributes to the result in such algorithms means that search enhancement techniques such as alpha-beta pruning cannot be used to improve efficiency without affecting the values computed.

Currently, we are aware of no correct algorithm for identifying optimal strategies against best defence other than the exhaustive strategy minimisation algorithm of §6.4. This algorithm overcomes both the phenomena of non-locality and strategy fusion by the simple expedient of examining the possible outcomes of each complete strategy separately.

6.7.1 A Bridge Example

We give below an example of non-locality as it can occur in Bridge. Note that it is difficult to construct simple examples for the non-specialist in Bridge. The example below is, however, representative of the kinds of problems that can arise during actual play.

Consider the situation of Figure 6.18, where one trick must be lost as the highest remaining card is held by the opposition. For ease of exposition we will assume that the possible options are represented using the tactics of Chapter 5, which specify not just a card to lead on a trick, but also the card to played by the declarer from the third hand, after the first opponent has responded.¹

We will look at the two tactics which succeed under most worlds. The first of these is

¹ Note that Ginsberg suggests a similar representation when discussing the example of Figure 6.12. When leading a card, he proposes that declarer should ‘decide in advance’ the other card he will play on the trick — essentially creating tactics. Within the repeated minimaxing framework, such a representation change has the effect of delaying the onset of strategy fusion for one level of search, since instead of choosing between individual moves — which are really just partial strategies with only the first step determined — we are now choosing between options which are partial strategies with the first *two* steps determined. However, the strategy fusion which results from allowing *different* completions of these partial strategies in *different* worlds will still remain beyond this new horizon. Although the use of tactics can indeed ‘correct’ the problems caused by strategy fusion in Ginsberg’s example, then, it cannot provide a complete solution unless the ‘tactics’ extend to the end of the play, at which point they become complete strategies.

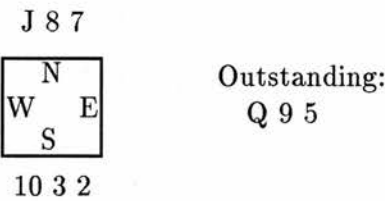


Figure 6.18: A Bridge card combination where one trick must be lost

a play which begins by leading the 7 from the North hand. If East plays the Queen straight away, declarer plays a low card from the South hand and wins the next two tricks with the Jack and the 10. If East plays the 9, declarer covers with the 10 and is again guaranteed two tricks. Finally, if East plays low, declarer again plays a low card from the South hand, hoping to find that East holds the 9, thus either forcing out the Queen or winning the trick outright. This play is a Type 2 finesse (of the 7) against East. The other possibility we will examine is a Type 3 of finesse, this time of the Jack, against West. This play begins by leading a low card from the South hand, intending to play the Jack unless West plays the Queen. Of the eight possible ways to split the outstanding cards, the distributions under which each of these tactics would produce two tricks are shown in Table 6.19.

Possible worlds: East holds	finesse of the 7	finesse of the J
Q95	•	•
Q9	•	•
Q5		•
95	•	•
Q	•	•
9	•	•
5		•
void	•	

Figure 6.19: Possible worlds under which each tactic will produce 2 tricks

With no further information to guide a choice, then, the most promising of the two tactics appears to be the finesse of the Jack. This will fail to win two tricks in only

one of the eight possible worlds (which is also the least likely), whereas the finesse of the 7 will fail in two. However, notice that the situation of Figure 6.18 can in fact be reached in one round of play (each player contributing one card) from the state where the cards are initially distributed as in Figure 6.20.

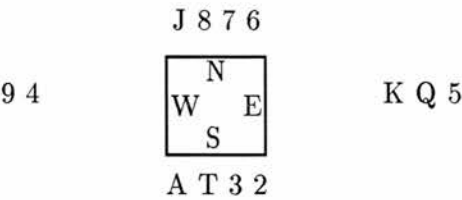


Figure 6.20: A Bridge example giving rise to non-locality

If the declarer does not know the actual distribution of the outstanding cards, his best play in this situation is to lead the 6 from the North hand and play low from the South hand unless East plays the 9 or one of the King or Queen. Faced with this play, East's best option if he holds the cards shown is to play low with the 5. If he does this, West will win the trick with the 9 and the declarer will be restricted to just two tricks in the suit. Similarly, if East starts with Kx or Qx, the best option will again be to play low. Thus, if the situation of Figure 6.18 is encountered as the result of leading the 6 and having to beat the King with the Ace, there will be extra information about the lie of the cards. Specifically, the cases where East holds the Q5 or the singleton 5 can be ruled out, because under these circumstances another branch, *higher* in the tree, would have been chosen by East to restrict the declarer to just two tricks.

This extra information would have a crucial effect on the situation in Figure 6.18. If it was known that this position would *not* be reached under the two distributions where East holds the Q5 or the 5, we can see from Figure 6.19 that the finesse of the 7 would then succeed under *all* the remaining worlds, whereas the finesse of the Jack would still fail in one. The probability of the finesse of the 7 producing two tricks in this scenario would therefore be higher than that of the finesse of the J, and the selection made in this position would need to be reversed.

6.8 Summary

We have looked at the problem of selecting a strategy in zero-sum two-player games with incomplete information. We formalised the notion of a best defence model of such games in which equilibrium point strategies could be more easily identified, and presented exhaustive strategy minimisation as an algorithm capable of finding such strategies.

Although the problems we address seem central to the design of programs for playing games in incomplete information domains, it appears that they have not been adequately studied or addressed in the literature. We were therefore able to newly formalise two general problems which can result from the incorrect handling of the technical notion of a strategy in such domains. The first of these, strategy fusion, results from combining different MAX strategies in different possible worlds. The second, non-locality, results from examining only partial strategies at internal nodes of a game tree.

We demonstrated that both strategy fusion and non-locality are present when the standard minimax algorithm is applied to games with incomplete information, as in the repeated minimaxing architecture. In the domain of Bridge, where we showed that the best defence model of the game corresponds to the typical model analysed in expert texts, our results must therefore call into question some of the claims about the prospects of computer players based on fast double-dummy programs. For instance, it seems at best unproven that such programs will inevitably mean that, as humans, ‘our time as world Bridge champions is limited’ [Ginsberg 95].

We pointed out that the main limitation of our model is the restriction to pure, rather than the general case of mixed, strategies. Also, our solution of exhaustive strategy minimisation, although capable of producing optimal strategies, is highly intractable. This suggests that there is a real cost involved in overcoming the effects of incomplete information. However, we have not yet established a lower bound for the complexity of this problem, and this remains an open — and apparently important — problem for the design of programs which play games like Bridge.

Chapter 7

Identifying The Best Strategy: Tackling Non-locality

‘What kind of place is Expectations?’ enquired Milo, unable to see the joke, and feeling very doubtful of the little man’s sanity.

‘Good Question, good question,’ he exclaimed. ‘Expectations is the place you must always go to before you get to where you’re going. Of course, some people never go beyond Expectations, but my job is to hurry them along whether they like it or not.’

— NORTON JUSTER
The Phantom Tollbooth

Baldric, you wouldn’t recognise a subtle plan if it painted itself purple and danced naked on a harpsichord singing ‘subtle plans are here again’.

— EDMUND BLACKADDER

In the previous chapter, we showed that the presence of incomplete information in a game gives rise to non-locality. We also showed that the solution of exhaustive strategy minimisation will only be feasible on the smallest of game trees. In this chapter, we therefore examine alternative ways in which the effects of non-locality can be dealt with. However, we have already noted that we know of no *algorithm* which identifies guaranteed optimal strategies more efficiently than exhaustive strategy minimisation. The approaches we discuss here will therefore to a greater or lesser extent be heuristic in nature.

Initially, we introduce the notion of the *qualitative* representation of incomplete information (§7.1), and discuss how such an approach can improve efficiency. In §7.2 we

then look at how the shortcomings of local evaluation functions can be addressed by providing them with extra information about the rest of the tree. Finally, we apply these ideas in the game of Bridge (§7.4 and §7.5).

7.1 Representing Information Qualitatively

The exhaustive strategy minimisation algorithm examines each possible MAX strategy under each possible outcome of the chance moves in a game. Thus, when the algorithm is applied to a game tree, large portions of the tree will be examined more than once. In order to reduce the number of times each node is examined, information can be represented *qualitatively*.

7.1.1 A One-pass Approach

Let us refer back to the game tree we introduced in the previous chapter as Figure 6.1 on Page 109, and again consider its flattened form. Rather than labelling the leaf nodes with payoffs in each world separately as we did before, however, let us use a single vector, \vec{K} , with one element for the payoff in each of the five possible worlds. In Figure 7.1 these vectors are represented by ovals which contain the possible payoffs in worlds w_1, w_2, w_3, w_4 and w_5 .

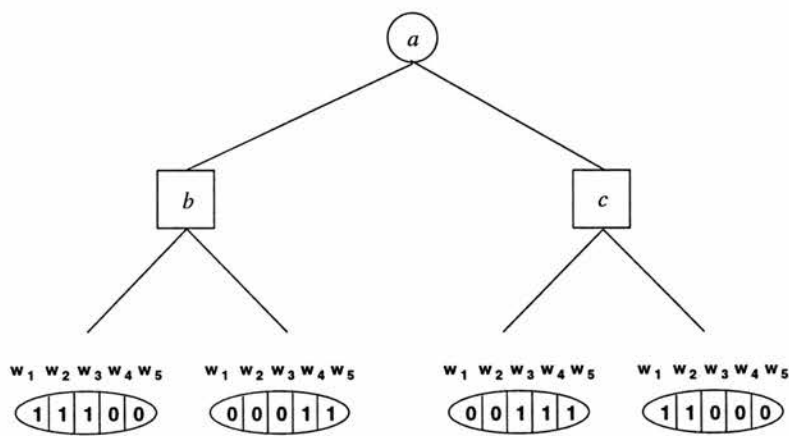


Figure 7.1: Representing payoffs as ‘vectors’ of possible outcomes

It is possible to analyse this type of tree with an algorithm which produces the same res-

ults as exhaustive strategy minimisation, but achieves this with just one pass through the tree. To do this, instead of selecting a single branch at each MAX node, the result of *each* possible selection is calculated and stored. For instance, in our example above, let us label the payoff vector at each leaf node with a template to represent a *partial strategy* which will become instantiated as the tree is processed (see Figure 7.2). At node *b*, MAX has two choices (either the left or the right-hand branch). We therefore raise the two vectors which are at the leaf nodes and store them at node *b*. Also, since *b* is the first node that will be encountered in a pre-order traversal of the tree, the two partial strategy templates are filled in correspondingly to indicate which vector is produced by selecting the left-hand branch, and which is produced by selecting the right.

Similarly, there are two possible vectors for MAX to select between at node *c*.

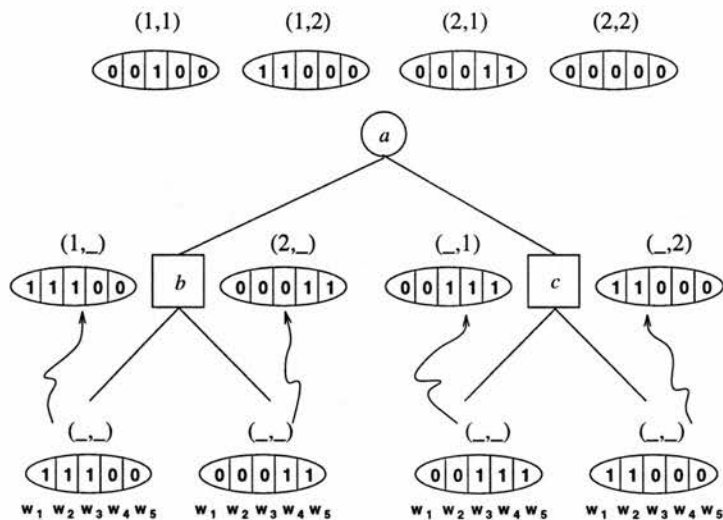


Figure 7.2: Annotating each node with a ‘vector’ describing the outcomes of the possible strategies

At node *a*, we have to analyse MIN’s possible actions in response to each of the possible combinations of choices that MAX may make at nodes *b* and *c*. This can be done by looking at each of the possible combinations of the vectors and partial strategies that label the MAX nodes. For example, let us consider how to combine together the choices represented by the partial strategies *(1,-)* and *(-,1)* (corresponding to choosing the left-hand branch at nodes *b* and *c*). MIN’s desire to minimise MAX’s payoff is modelled by comparing the payoffs in each of the associated vectors and selecting the smallest

payoff in each possible world. This results in a new vector with a single 1 in world w_3 . The identification of the strategy represented by this vector is found by combining together the partial strategies to produce (1,1). Thus, we can now label node a with an oval representing the payoff for this strategy. This procedure is repeated for the other possible combinations of the strategies at nodes b and c , producing annotations at node a for the strategies (1,2), (2,1) and (2,2), as shown in Figure 7.2. These are the same results as those obtained by the exhaustive strategy minimisation algorithm described in the previous chapter. That is, strategy (1,1) is found to give a payoff of 1 in just world w_3 , strategy (1,2) is found to give a payoff of 1 in just worlds w_1 and w_2 , etc. Thus, we have analysed all the possible strategies whilst examining each node of the tree only once.

7.1.2 Discussion

The exact formalisation of an algorithm using qualitative information will depend on the method used to represent and build the strategies. Since doing this efficiently can be complicated, we omit a formal specification and concentrate instead on the general benefits to be gained from a one-pass algorithm.

Of course, analysing the tree in one pass does not of itself imply greater efficiency, as there is a penalty incurred by having to maintain many vectors at each node. For a general tree, the number of the vectors present at any MAX or MIN node is given by the doubly exponential recurrence relations of §6.4.5. Thus, whereas the $esm(\Gamma)$ algorithm calls $esm(t,s)$ once for each possible strategy, the approach outlined above has to retain the results of each traversal of each subtree at its root. Also, the time complexity of such an algorithm on a general tree will be of the same order as that of the exhaustive strategy minimisation algorithm, since for each MAX strategy the result in each of the possible worlds is eventually examined.

However, there are two ways in which this basic one-pass algorithm could be improved to enhance its performance. Firstly, it is clearly true that if the collection of vectors at a node contains members which are pointwise less than or equal to any other member at that node, these vectors may be ignored as inevitably giving rise to inferior strategies. For example, the vector containing entirely zeros at node a may be omitted from further

consideration, as there are other vectors at the same node that offer at least as good a payoff in every world. Prunings of this sort may lead to a noticeable improvement in performance, but the effect will vary from domain to domain and from problem to problem.

A more important improvement lies in an appropriate choice of representation for the payoff information at each node. Rather than representing each payoff in a single slot of a vector, it is possible to group possible worlds together in a *qualitative* representation. For instance, in our current example, consider the effect of using a simple propositional language to group together the worlds in which the payoffs are identical (see Figure 7.3). The generation of the vectors at the root of the tree is now conducted

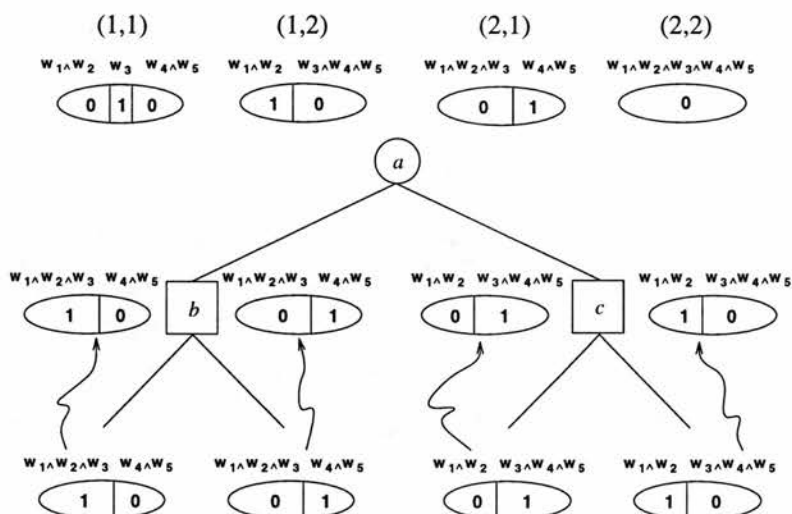


Figure 7.3: Improving efficiency via the use of a simple propositional language

not by comparing the payoffs in each possible world, but by manipulating the actual propositional expressions. Thus, by a suitable choice of representation language, the complexity that results from the requirement to examine the results of each strategy in each possible world can be reduced; the qualitative representation allows worlds with the same payoff to be processed together. In §7.4, we describe how FINESSE uses a qualitative language of simple predicates to describe the uncertainty in Bridge. Typically, this replaces the task of considering all the possible worlds with that of manipulating an expression with a very small number of terms. However, the degree of improvement

offered by this kind of representation change is again difficult to quantify in general. In the next section, we therefore look at the prospects for more general techniques.

7.2 Parameterised Local Evaluation Functions

We know that in game trees with incomplete information, non-locality makes it impossible to choose the correct action at any internal node of the tree by just analysing the node's subtree. The exhaustive strategy minimisation algorithm, and the one-pass approaches described above, avoid making local evaluations but pay the penalty of having to examine large amounts of information. Here we will examine an alternative approach: that of using local evaluations parameterised by some extra information about the situation in the remainder of the tree.

7.2.1 Doublethink

Consider the algorithm of Figure 7.4, which for reasons we describe below, we call *naïve minimaxing*. This algorithm returns strategies under the assumptions of the best defence model for the flattened form, t , of a game with payoff vectors at the leaf nodes, \vec{K} , as described above. This algorithm commits to just *one* choice of branch at each MAX node, while allowing MIN to select any branch in any world.

Algorithm naïve-mm(t): Take the following actions, depending on t .

Condition	Result
t is leaf node	$\vec{K}(t)$
$\mathcal{P}(\text{node}(t))$ is 2 (i.e., MIN to move)	$\min_{t_i \in \text{sub}(t)} \text{naïve-mm}(t_i)$
$\mathcal{P}(\text{node}(t))$ is 1 (i.e., MAX to move)	$\max_{t_i \in \text{sub}(t)} \text{naïve-mm}(t_i)$

Figure 7.4: Naïve form of a minimax algorithm on flattened trees

Here, the normal min and max functions are extended so that they are defined over a set of payoff vectors. At a node with m branches and therefore m payoff vectors

$\vec{K}_1, \dots, \vec{K}_m$ to choose between, we interpret the min function as:

$$\min_i \vec{K}_i = (\min_i \vec{K}_i[1], \min_i \vec{K}_i[2], \dots, \min_i \vec{K}_i[m]), \quad (7.1)$$

where $\vec{K}_i[j]$ is the j th element of the vector \vec{K}_i . That is, the min function returns a vector in which the payoff for each possible world is the lowest possible. This is equivalent to allowing MIN to choose any branch under any possible world.

MAX, however, does not have knowledge of the possible worlds, so the max function returns the single vector with the highest expected payoff. That is, in a game with N possible worlds, it returns the vector, \vec{K}_i , for which

$$\sum_{j=1}^N \Pr_{max}(\mathbf{w}_j) \vec{K}[j] \quad (7.2)$$

is maximum, where $\Pr_{max}(\mathbf{w}_j)$ represents MAX's assessment of the probability of the actual world being \mathbf{w}_j .

We can see from these equations how the problem of non-locality arises. In (7.1) MIN can choose different branches in each world. However, since the algorithm is recursive, the MIN choices at any node will always be made *after* MAX's choices in the subtrees of that node (in the same way that MIN chooses after MAX in the best defence model of a game). When analysing any MAX node, then, the choices to be made at its MIN ancestors will not yet have been decided, and the possible worlds under which the play will reach the node will therefore be unknown. MAX's evaluation in (7.2) ignores this problem, simply taking the *naïve* approach of assuming that *every* world is possible.

To cope with this problem, we can imagine that each time MIN's evaluation function selects a branch as the best MIN move in world \mathbf{w}_j , the alternative branches which will *not* be selected (*i.e.*, those with higher evaluations in this world), are 'annotated' with the label $\neg \mathbf{w}_j$. For any internal MAX node, the subset of possible worlds under which it will be reached can then be found by excluding all the worlds annotated on the MIN branches which occur in the path from the node to the root.

Non-local information could therefore be tackled with a two-phase algorithm which

attempts to use such inferences to improve MAX's move selections. Whilst carrying out an initial naïve minimaxing of the game tree, we can annotate each MIN branch with the worlds under which MIN would *not* select it, as described above. Then, a second pass through the tree can be made, in which these annotations are taken into account when making selections at MAX nodes. MAX's new formula for choosing a vector at a node where \mathcal{W} represents the *exclusion set* of the indices of the possible world annotations in the path connecting it to the root node would then be:

$$\sum_{\substack{j=1 \\ j \notin \mathcal{W}}}^N \Pr_{\max}(\mathbf{w}_j) \vec{K}[j]. \quad (7.3)$$

Thus, MAX can attempt to make inferences about the plays that would be selected by an optimal MIN. These inferences may alter MAX's branch selection at any node and maybe his overall prospects in the game. They address the problem of non-locality by allowing MAX to infer some information about the possible situations under which any node may be reached. The process can thought of as conducting an extra reasoning step which is akin to *doublethinking* MIN. As an example, consider the flattened tree of Figure 7.5. In this tree, there are four possible strategies, with payoffs as shown in

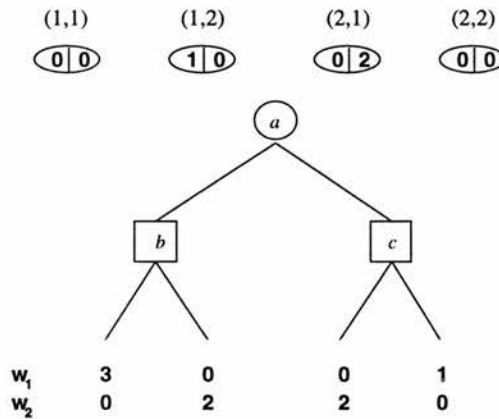


Figure 7.5: Simple example of a flattened tree

the figure. The best of these (given equally likely worlds) is strategy (2,1). However, if MAX simply uses the naïve algorithm of Figure 7.4, he will select the left-hand branches at nodes b and c , which corresponds to choosing strategy (1,1). To modify

this choice, we could use doublethink to make inferences about which branches will be selected by MIN. This will lead MAX to believe that he can improve his chances, and to change his selection at both nodes as shown in Figure 7.6. Since we do not allow MIN to change his branch selections after this doublethink stage, MAX will expect a payoff of 1 in world w_1 and a payoff of 2 in world w_2 .

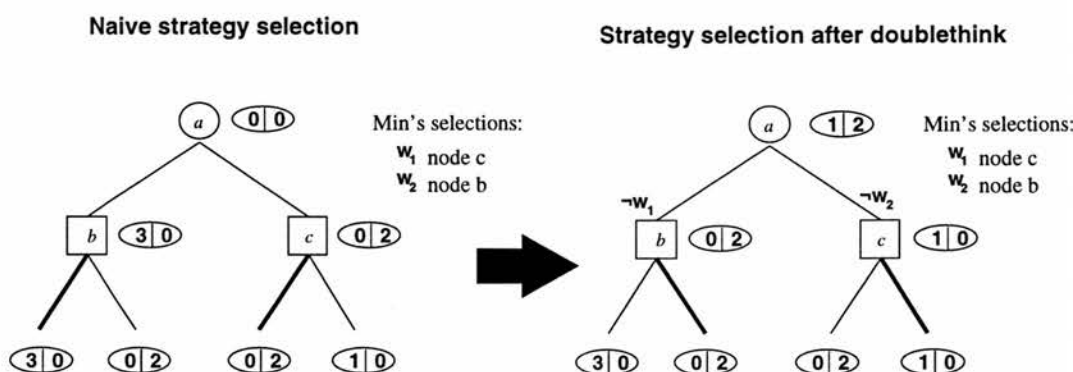


Figure 7.6: Carrying out doublethink

7.2.2 Oddthink

Of course, if we allow MAX to doublethink MIN, we have in fact made MAX's strategy selection conditional on both the local evaluation of a node and the normal strategy that MIN would select. Thus, the best defence assumption A-II that MIN chooses his strategy after MAX is violated. In order to correctly model the best defence assumptions, we therefore need to add another layer of reasoning by taking the MAX strategy resulting from the doublethink step and allowing MIN to re-evaluate the best choices at each MIN node — a step which could be described as *triplethink*.

To see that MIN's choices may change when applying triplethink, consider again a MAX node at which, in the naïve algorithm, MAX finds the best choice to be branch a . If this is the case, then

$$\forall i. i \neq a \Rightarrow \sum_{j=1}^N \vec{K}_a \Pr(w_j)_{max} > \sum_{j=1}^N \vec{K}_i \Pr(w_j)_{max}. \quad (7.4)$$

Let us say that when conducting doublethink, however, the exclusion set for the node is found to be \mathcal{W} , and the move which achieves the maximum evaluation (using the modified assessment produced by (7.3)) changes to branch b , *i.e.*:

$$\sum_{\substack{j=1 \\ j \notin \mathcal{W}}}^N \vec{K}_a[j] \Pr_{max}(\mathbf{w}_j) < \sum_{\substack{j=1 \\ j \in \mathcal{W}}}^N \vec{K}_b[j] \Pr_{max}(\mathbf{w}_j). \quad (7.5)$$

For this to be true, $\vec{K}_a[j] < \vec{K}_b[j]$ must hold for at least one value of $j \notin \mathcal{W}$. And, if this and (7.4) are both to be true, $\vec{K}_a[j] > \vec{K}_b[j]$ must be true for at least one value of $j \in \mathcal{W}$. Therefore, MAX's doublethink has the effect of reducing the payoff in at least one of the worlds represented by the exclusion set, and increasing the payoff in one of the remaining worlds.

Now, if it happens that the reduction of the payoff for a world in the exclusion set takes its value below the previous minimum to which MIN could restrict MAX at that node, allowing MIN an extra step of reasoning *will* lead him to go down the branch in this world, despite MAX's inference that he will not. MAX's doublethink inferences, therefore, can be used by MIN to actually *reduce* his payoff in certain worlds, if triplethink is allowed.

Similarly, the increase in payoff for a world not in the exclusion set may be a false improvement for MAX, since if the increase takes the payoff above the minimum payoff of its siblings, MIN will no longer select the node in this world, preferring instead to restrict MAX's payoff by selecting one of the other branches. Thus, if MIN is allowed to conduct another level of reasoning, the worlds present in the exclusion set of a node may no longer be correct. Also, worlds which should be present may be absent. To remedy this, and to be sure of making the best moves, MAX would need to conduct another level of reasoning about MIN's strategy.

In general, however, it is not easy to see where this chain of reasoning will terminate. It is in fact relatively simple to construct situations in which successive levels of reasoning will loop forever through a finite set of choices. For instance, let us return to our example of Figure 7.5. In this example, we allowed MAX to use doublethink to change his strategy. But benefit for MAX is also disadvantage for MIN, and to model the

best defence assumptions correctly we have to allow another stage of reasoning. In this triplethink step (see Figure 7.7), the return which MAX was expecting from his new strategy is revealed to be over-optimistic; instead of achieving a 1 in w_1 and a 2 in w_2 , he achieves zero in both. Two more stages of reasoning then bring us back to the same situation as in the original naïve scheme.

It is possible to imagine an algorithm that adapts this approach, conducting a fixed number of ‘thinking’ steps (possibly with a check for looping) and then selecting the best strategy from amongst those encountered. In order to correctly model the best defence assumptions, however, the last step in this chain must be a MIN reasoning stage. Thus, since ‘even’ levels of reasoning (doublethink, 4-think, *etc.*) are carried out by MAX, and ‘odd’ levels by MIN, such an approach could be described as an *oddthink* algorithm. However, notice that our example demonstrates that the process of adding extra reasoning steps may terminate *before* the optimal strategy is examined; the two strategies encountered are (1,1) and (2,2), which are both inferior to the two strategies that remain.

7.2.3 Payoff Reduction

There are in fact two types of inferences involved when producing the exclusion set of a node as described above. The annotations which were made in Figure 7.7 were found to *change* each time MIN was allowed an extra reasoning step. However, there are some inferences which MAX can make which will *never* be invalidated by future reasoning carried out by MIN. For example, consider the flattened tree of Figure 7.8 on Page 157, in which we have added one extra possible world to our previous example.

In this tree, the payoffs of 8 and 7 in world w_3 will *never* be realised, because the two zeros under node c will always enable MIN to restrict MAX to a zero in this world. Thus, the addition of the extra possible world has no effect on the assessment of the best strategy, since node b will *never* be reached under world w_3 , whatever strategy is selected by MAX. We can therefore annotate node b with the term $\neg w_3$, in the knowledge that this inference will never be incorrect. This inference is different in nature to the ones that we saw in the previous section, whose truth depended on the actual strategy chosen by MAX. For instance, if we look again at node b we can see

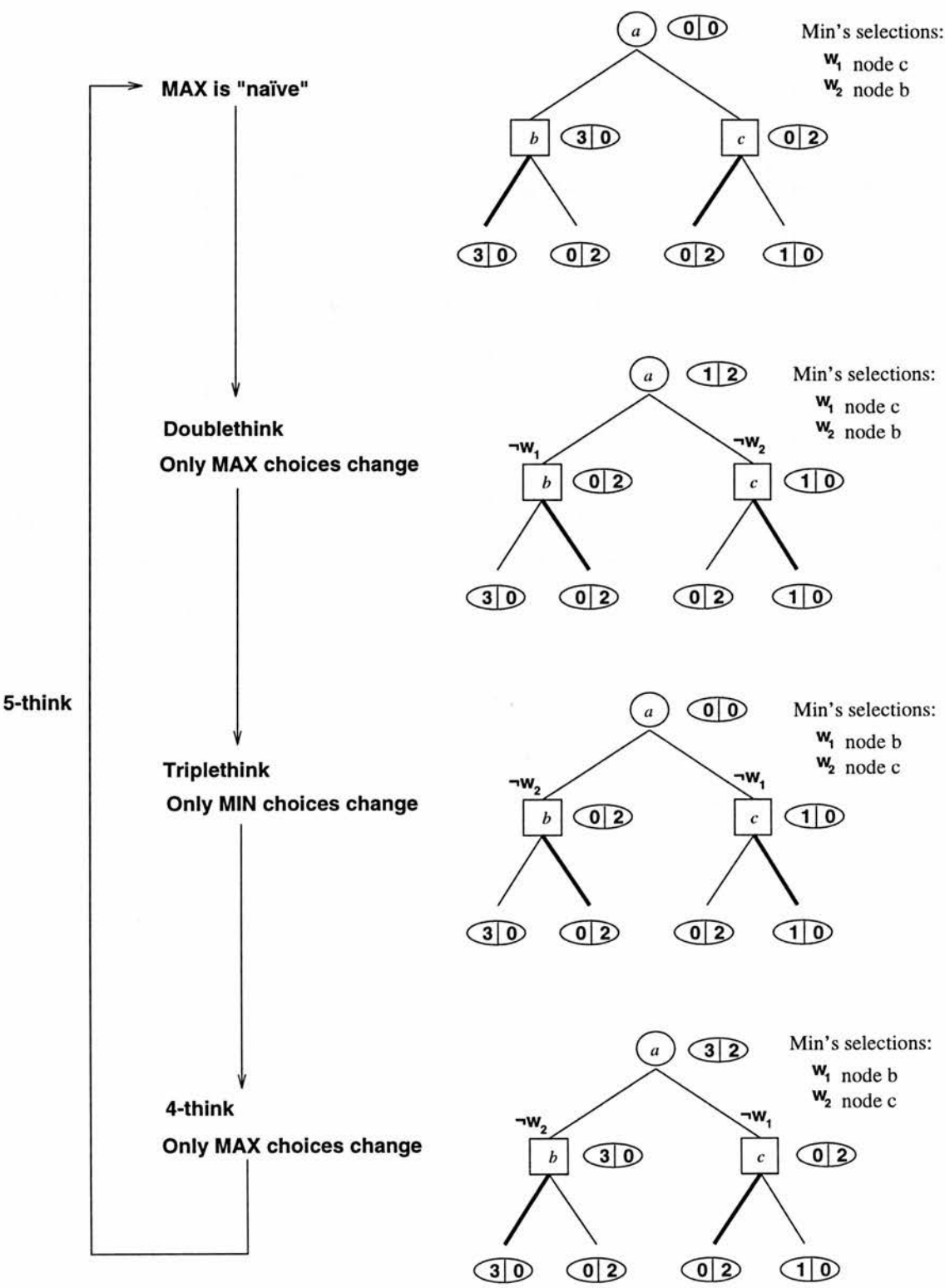


Figure 7.7: Simple example of how multiple levels of reasoning gives rise to loops

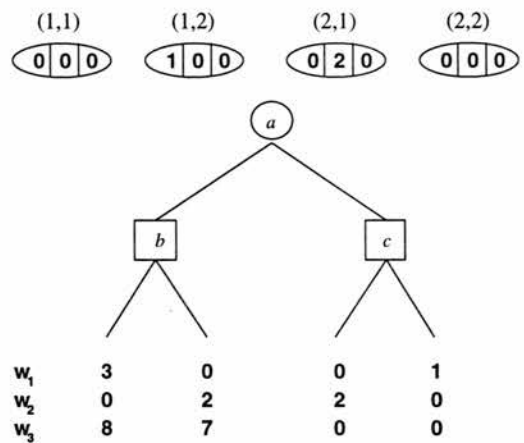


Figure 7.8: Flattened tree example with one extra world added

that if MAX selects the left-hand branch here, MIN’s evaluation function will always back up the value from node c in world w_1 . This fact should be represented by a *conditional* inference at node b such as ‘MIN will not direct play down this branch in world w_1 if the left-hand branch is selected’. The $\neg w_2$ annotations in Figure 7.7 are also conditional on the strategy selected by MAX.

It is possible to envisage a two-step algorithm which first identifies all the *unconditional* inferences which are correct independent of strategy selection and then uses these to supply extra information to a local evaluation function. The use of inferences that are always correct would make it unnecessary to allow MIN a further level of reasoning after MAX’s doublethink step. However, it is possible to do better than this.

Consider what happens when a branch is selected at some MAX node. The need to make inferences arises when the payoff in some world on that branch is higher than the payoff in the same world at a branch selected at one of its siblings. For, when this happens, we can be sure that MIN will direct the play towards the sibling node with the lowest payoff in that world. The *actual* values that are achieved at the sibling nodes depend on the strategy selection made by MAX. However, here we can apply some reasoning about the nature of the game. If MAX had exact knowledge of the state of the world, and the game was played perfectly as a complete information game, the best that he could hope for would be to obtain a payoff equal to the minimax value

in that world. Therefore, in any world, the complete information minimax value of the game under that world forms an upper bound on MAX's payoff in that world (since MAX does *not* know the state of the world and MIN does). Thus, when making a branch selection at a node, any payoff which exceeds this upper bound in some world will never be reached in that world. Such payoffs may adversely affect branch selection at the node. Rather than checking payoffs against the *actual* values at sibling nodes, then, we can compare them to the minimax value of the game in each world.

In fact, this argument holds not just for the minimax values of the entire game tree, but for every MIN node ancestor; any branch selection that MAX makes in anticipation of a payoff higher than the minimax value of a MIN ancestor is bound to be based on false information, since on the way to this MAX node, play must pass through a node at which MIN will always be able to direct the play towards a lower payoff. The conclusion to be drawn, then, is that if we select a branch with a payoff higher than the minimax value of any of its MIN ancestors in some world, MIN will avoid the node in this world.

A simple way to implement this observation is to modify the payoffs of any given game so that *none* of them exceed the minimax value of a MIN ancestor in any world. We can achieve this with the *payoff reduction minimizing* algorithm of Figure 7.9. The

Algorithm *payoff-reduce-mm*(*t*):

Returns strategies for flattened game trees under the best defence model:

1. Find the minimax values of the nodes of the tree in each possible world.
2. Examine the payoff vectors of each leaf node. Those that exceed the minimax value of any of their MIN ancestors in the corresponding world are reduced to this value, and those that do not are left unchanged.
3. Apply the *naïve-mm* algorithm to the resulting tree.

Figure 7.9: The payoff reduction minimizing algorithm

reductions of the payoffs made by this algorithm do not affect the minimax value of the tree in each world, since no payoff is reduced to the extent that it would offer MIN a better branch selection at any node in any world. For example, let us consider how the algorithm would behave on the previous tree of Figure 7.8. The minimax values in

each of the three possible worlds in this game are 1, 2, and 0, respectively. Thus, we begin by reducing all the payoffs in each world to at most these values (there is only one MIN node in the game, so each leaf node has only one MIN ancestor). This is shown in Figure 7.10, where the strategy selection subsequently made by the *naïve-mm* algorithm has also been highlighted in bold. In this tree, then, the payoff reduction algorithm results in the correct strategy being chosen.

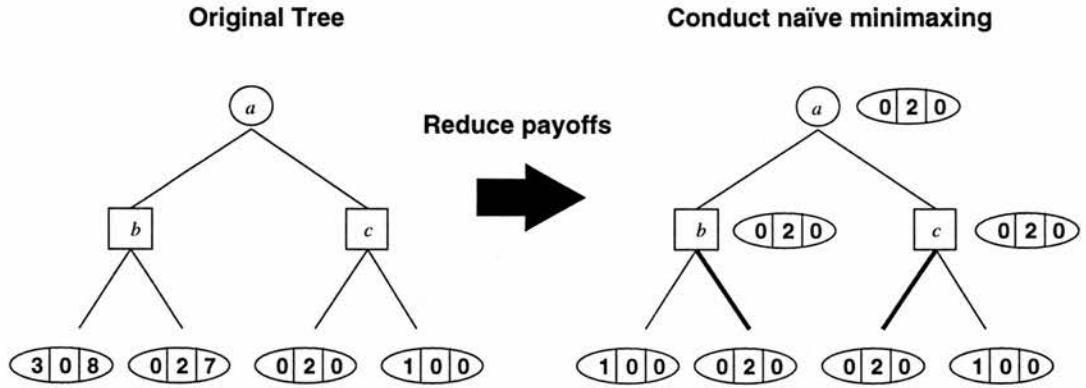


Figure 7.10: Applying the *reduce-payoff-mm* algorithm

It may seem that this algorithm is inefficient, since it has to conduct minimaxing once in each world, then reduce the payoffs, and finally carry out naïve minimaxing. However, it is possible to make some of these steps more efficient and to combine others together. For example, the minimax value of the game in each possible world can be calculated concurrently in one pass of the tree. This can be done, for example, by using a modified form of the *naïve-mm* algorithm in which MAX has the same luxury as MIN to choose different branches in different worlds. That is, we can rewrite the definition of the max operator over a set of payoff vectors to be:

$$\max_i \vec{K}_i = (\max_i \vec{K}_i[1], \max_i \vec{K}_i[2], \dots, \max_i \vec{K}_i[m]). \quad (7.6)$$

In Figure 7.11 we apply this modified algorithm to our example tree. At node *b*, MAX uses (7.6) to choose the payoff in each world and finds that payoffs of 3, 2, and 8 can be achieved. Similarly, a vector containing payoffs of 1, 2, and 0 is produced at node *c*. Then, MIN applies (7.1) to build the vector of minimax values at the root.

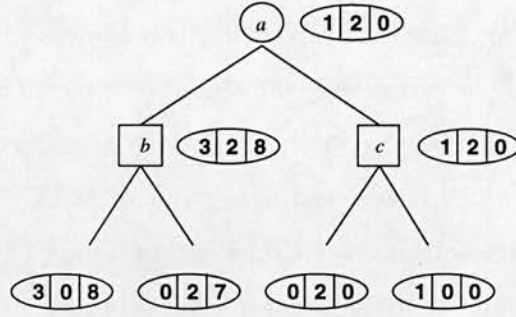


Figure 7.11: Conducting minimaxing in each world concurrently

Calculating minimax values in this way allows for the possibility of efficiency improvements through the use of a qualitative representation. A re-modelled *payoff-reduce-mm* algorithm incorporating this approach is given in Figure 7.12.

Algorithm *payoff-reduce-mm*(t):

Returns strategies for flattened game trees under the best defence model:

1. Conduct (qualitatively) minimaxing on single worlds, storing at each MIN node its minimax value in each world.
2. Apply a modified form of *naïve-mm* to the annotated tree: since this stage can be implemented depth-first, keep a record, for each world, of the minimum annotated value encountered at the MIN nodes leading to the current position in the tree — when encountering leaf nodes, reduce the payoff vector \vec{K} to at most these values in each world.

Figure 7.12: Modified form of the payoff reduction minimaxing algorithm

Notice that the *payoff-reduce-mm* algorithm makes all the unconditional inferences we discussed at the beginning of this section. These inferences occur when a node will *never* be visited in a particular world; that is, when the payoff in that world on *every* branch at the node is higher than the minimax value of a MIN ancestor. For example, consider world w_3 in our example tree. At node b , both the payoffs in this world are higher than the minimax value of the tree (which is zero). The reduction of these payoffs to this value is effectively the same as ignoring the world altogether, given equally likely worlds.

However, *payoff-reduce-mm* still does not make *all* the inferences which are possible,

since it is probable that MAX will not be able to achieve the actual minimax value of the tree in all worlds simultaneously. When this is the case, it is possible that some MAX node selections will be erroneously influenced by a payoff in some world which although not higher than the minimax value of any MIN ancestor in that world, is higher than the actual value which is achieved as a result of MAX's other branch selections. For example, in the tree of Figure 7.1 on Page 146, payoff reduction has no effect, since the minimax value of the tree under every world is 1. On this tree, then, the *payoff-reduce-mm* algorithm selects the same (incorrect) strategy as the simple *naïve-mm* algorithm. Nevertheless, *payoff-reduce-mm* was the best general algorithm we were able to find for tackling the effects of non-locality more efficiently than exhaustive strategy minimisation.

7.3 Application to Bridge: The Interpreter Algorithm

So far we have looked at the difficulties of tackling non-locality in *general* game trees. Now, we move on to consider how the problem can be dealt with in the game of Bridge.

In fact, we found that in Bridge, the use of high-level tactics to construct the tree of possible moves in each suit allowed us to produce a simple heuristic to counter some of the effects of non-locality. This heuristic allows certain responses to some tactics to be discounted as plays which the defence should *never* make. Therefore, FINESSE itself does not make use of the doublethink or the payoff reduction ideas discussed above, but instead uses a version of the qualitative naïve minimaxing algorithm modified to take account of these heuristic inferences. This algorithm forms the *interpreter* algorithm first mentioned in the system picture given in the introductory chapter. The basic function of the interpreter is to take the tree of legal play sequences produced by the single-suit planner and identify the best branch selections at each MAX node. In §7.4 we look at the qualitative representation language which makes it possible to do this efficiently, and in §7.5 we discuss the heuristic for tackling non-locality. Below, we begin by describing how the interpreter algorithm instantiates the basic *naïve-mm* framework.

7.3.1 Action at Leaf nodes

At each leaf node of the tree a term is generated which represents the possible payoffs qualitatively. These *payoff profiles* are lists of pairs (which we conjoin by a \triangleright sign for the sake of readability) of the general form

$$[T_1 \triangleright C_1, T_2 \triangleright C_2, T_3 \triangleright C_3, \dots] .$$

Here, the T_i are distinct numbers of tricks, in ascending order, and the corresponding C_i are terms which describe the possible worlds under which this number of tricks will be taken. Since this uncertainty representation language is complex, we describe it in §7.4, giving it a section of its own. Here, we concentrate on the details of the interpreter algorithm itself.

7.3.2 Action at Internal Nodes

For any non-leaf node, let us say that its i th daughter is described by the payoff profile K_i :

$$K_i = [T_{i1} \triangleright C_{i1}, T_{i2} \triangleright C_{i2}, T_{i3} \triangleright C_{i3}, \dots] . \quad (7.7)$$

We can then define the max and min functions from the *naïve-mm* algorithm to operate on these profiles as follows.

MIN nodes

At any MIN node, MIN's ability to select different branches in different possible worlds is modelled by defining $\min_i K_i$ to be the list of pairs of the form $T \triangleright C$, where T varies over each possible value taken by the T_{ij} and the corresponding C is the disjunction of all the C_{ij} for which $T_{ij} = T$.

If any world is described by more than one C in the resulting payoff profile, we assume the defence will restrict MAX to the lowest possible number of tricks. To model this, the initial profile constructed at the node is processed to remove any overlap in the C

terms. To do this, we start with the smallest payoff, $T_1 \succ C_1$, and make the set of worlds that C_1 describes as large as possible by removing any overlap with the remaining C_j . This process is then repeated for the remaining entries in ascending order. An example of this where there are just three possible payoffs is shown in Figure 7.13. We refer to this process as *subsumption*, and it is described in more detail in §7.4.4.

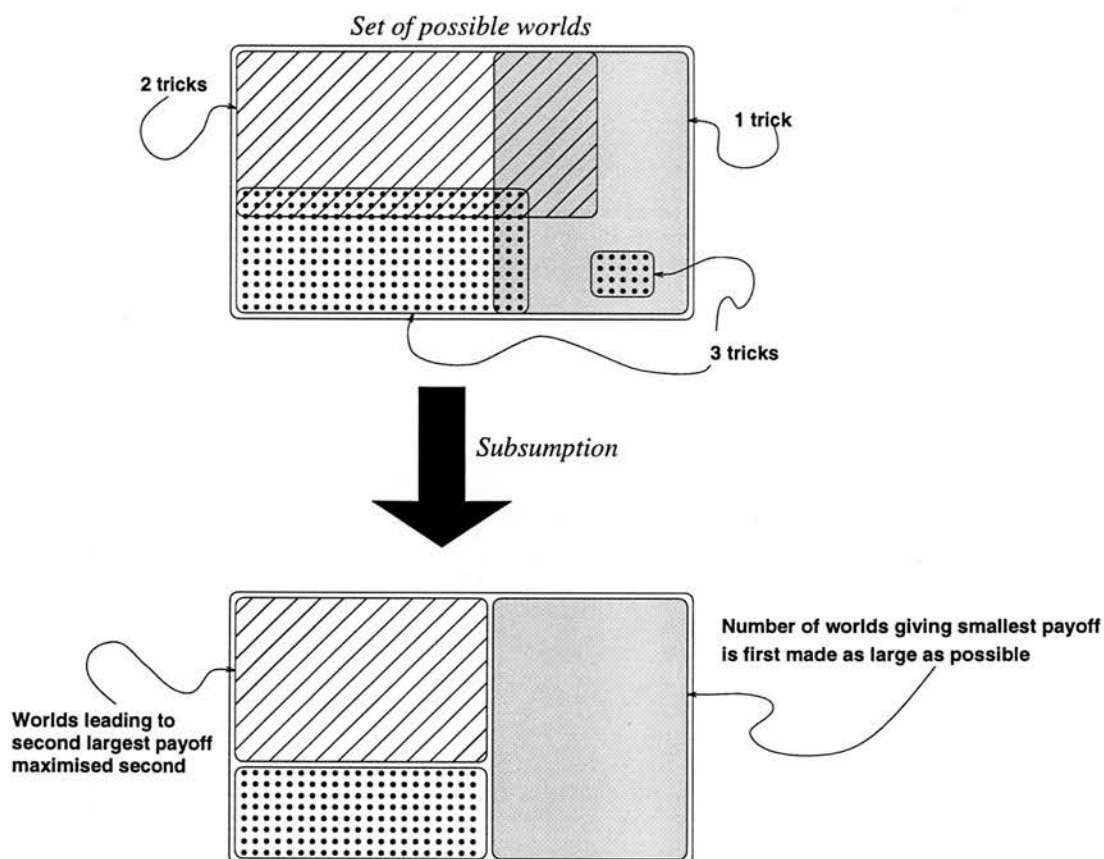


Figure 7.13: The use of subsumption to model minimisation of the possible payoff

MAX nodes

At any MAX node, the basic payoff profiles are converted into *probability profile* by finding the probabilities of each of the terms C_i (a technique for this probability generation is described in §7.4.6). The general form of such a probability profile is

$$[T_1 \succ p_1, T_2 \succ p_2, T_3 \succ p_3, \dots],$$

where the T_i are again distinct numbers of tricks, in ascending order, but this time the corresponding p_i are the probabilities of making each number of tricks. Given a set of these probability profiles, there are obviously a number of possible criteria by which a selection could be made. For example, the best expected value, the greatest chance of making a certain number of tricks, or the maximum chance of making the highest number of tricks all form possible bases for comparison. This is an issue which we will return to in Chapter 8, when we consider the task of combining together multiple plans from separate suits. For now, we will settle for the final option above of selecting the branch with the maximum chance of making the highest number of tricks, as this is a common goal when analysing single suits.

7.3.3 Lines of Play

When selecting a MAX branch, there may be more than one branch at which the maximum evaluation is achieved (according to whatever criteria is being used). If this is the case, and the payoff profiles of these branches are also identical, then *all* these branches are marked as selected. This is because such ‘equivalent’ branches typically differ just in the ordering of their actions, and retaining the ability to order the actions involved in any way will be a useful asset when we consider the process of forming global plans by combining together plans from individual suits. We use the term *line of play* to describe the tree of MAX selections which is produced in this way by the interpreter algorithm. It differs from the notion of strategy, in that a pure strategy forces a *single* selection in any information set. However, despite having multiple choices when following a line of play, the same payoff will always be achieved, irrespective of the choices made.

Below, we describe the uncertainty representation language which makes this interpreter algorithm possible by allowing efficient representation and manipulation of the possible worlds. In §7.5, we then describe the domain-specific heuristic that tackles the non-locality inherent in the naïve minimaxing approach.

7.4 Representing Uncertainty in Bridge

Here, we present FINESSE's qualitative uncertainty language, concentrating on the central issues of how terms in the language can be manipulated, and how they can be used to support the generation of probabilities. Notice, however, that in addition to allowing uncertainty to be represented efficiently, the use of a qualitative language offers other benefits, such as the ability to generate textual explanations or to use information deduced from the bidding phase to influence plan selection. FINESSE has only recently been coupled with a bidding system [Asher 93, Green 95] so it does not yet incorporate bidding inferences. However, in Appendix B we describe an implemented procedure for generating basic textual explanations (examples of the explanations generated by FINESSE can be found in Chapter 11 and in Appendix C).

7.4.1 Binary Strings

As we saw in Chapter 5, FINESSE groups all outstanding cards into consecutive sequences so that all the cards from any particular sequence can essentially be considered as equivalent. In general, then, the type of facts we are interested in describing are *how many* cards are held by one defender from any given outstanding sequence. To efficiently represent the disjunction involved in this task, FINESSE uses a representation based on binary strings.

The most basic situation the system deals with is that where one of the defenders holds $0, 1, \dots, n-1$, or n cards from an outstanding sequence of length n . For example, there may be an outstanding sequence in some suit consisting of the Ace, King and Queen. For any one player, there are eight possible ways to place these cards in his hand, as shown in Figure 7.14 on the following page, where we divide the possibilities into four groups according to the number of cards they contain.

To represent the possibilities for a general sequence of length n , FINESSE arbitrarily selects the perspective of one of the defenders (always East) and constructs a binary string of length $n+1$. Such strings can be utilised to represent a player's holding by using each bit in the string to specify whether East may hold a particular number of cards from the sequence. Specifically, FINESSE sets the bit corresponding to the i th

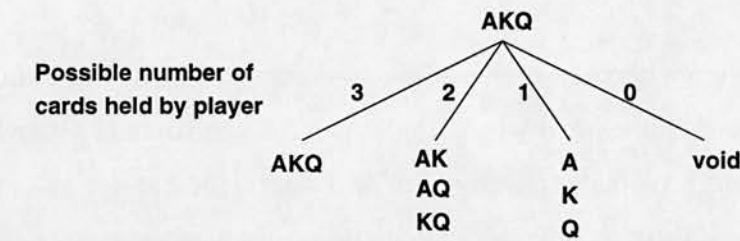


Figure 7.14: One player’s possible holdings from the sequence AKQ

power of 2 to 1 if East may hold i cards from the sequence, and 0 if he may not. For instance, Figure 7.15 shows how such strings can be used to represent some possible holdings from our example sequence AKQ.

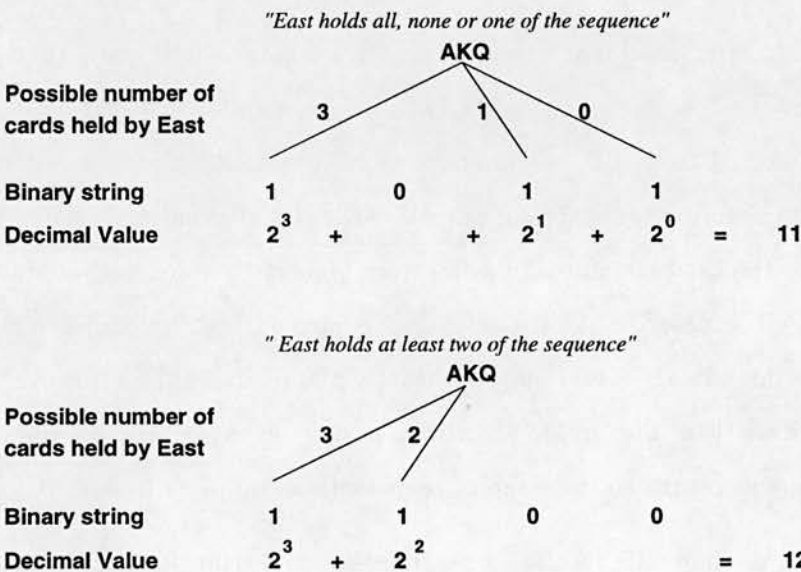


Figure 7.15: Examples of binary strings representing possible holdings from the sequence AKQ

The diagrams of Figure 7.14 and Figure 7.15 represent the disjunction of possible situations pictorially using single-level trees. We will pursue this analogy as we consider how compound expressions, describing multiple sequences of outstanding cards, are manipulated. First, however, we give some actual examples of such expressions.

7.4.2 \mathcal{C} -conjunctions

When there is more than one sequence of outstanding cards, binary strings representing East's possible holdings in each of the sequences are combined together to form a composite term. We have already seen in Chapter 5 that for the sub-problems of individual suits FINESSE distinguishes between the sequences of outstanding cards that are critical to the play of the suit and those which are not (the low cards). Thus, the general task in any given suit will be to represent the possible distributions of p sequences of outstanding critical cards and one sequence of low cards. To do this, FINESSE forms an expression containing $p + 1$ individual terms as follows:

$$\text{east}(\text{Bin}_1, \text{List}_1) \wedge \cdots \wedge \text{east}(\text{Bin}_p, \text{List}_p) \wedge \text{east_low}(\text{Bin}_{p+1}), \quad (7.8)$$

where each Bin_i is a binary string describing the possible cards that can be held by East from the corresponding card List_i (or from the low cards, if $i = p + 1$). Such a term is referred to as a \mathcal{C} -conjunction, since it consists of a series of conjuncts which most often describe sequences of critical cards. By convention, a \mathcal{C} -conjunction always represents the card sequences in order from highest to lowest. (\mathcal{C} -conjunctions can also be extended to describe the situation in a complete deal, by conjoining four single-suit \mathcal{C} -conjunctions in the order spades, hearts, diamonds, clubs. However, this is not an issue here, as the interpreter algorithm is only used to process plans for individual suits. We will return to the issue of representing complete deals in Chapter 8.)

Any \mathcal{C} -conjunction will describe a possible set of distributions of the outstanding cards between the two defenders. For example, consider the situation of Figure 7.16, where the King is the only critical card, and there are therefore five low cards. In Figure 7.17 we arrange East's possible holdings at the leaves of a tree. Any of the distributions in this tree can be simply represented using \mathcal{C} -conjunctions. For example, the left-most branch, where East holds all the cards becomes $\text{east}(10, [\text{king}]) \wedge \text{east_low}(100000)$. Notice also that a \mathcal{C} -conjunction may describe a *set* of distributions as well as just a single possibility. For example the term $\text{east}(11, [\text{king}]) \wedge \text{east_low}(001010)$ describes the distributions where either East or West holds the King, and East has one or three low cards. These are the distributions which we have

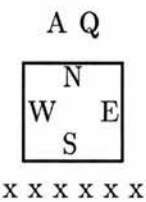


Figure 7.16: A simple card combination

represented with bold paths in the figure. Thus, \mathcal{C} -conjunctions can be thought of as representing subtrees consisting of collections of paths through the original tree of possibilities.

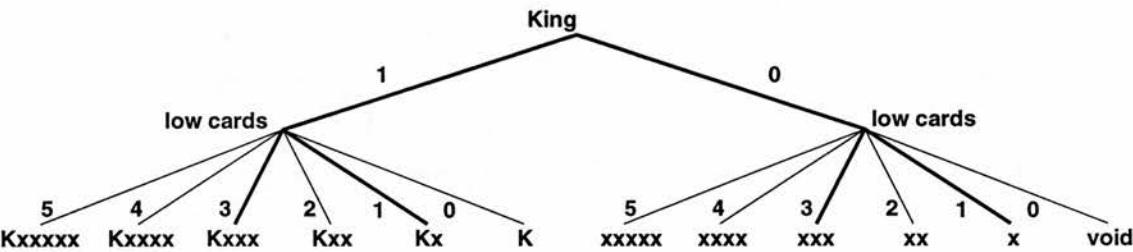


Figure 7.17: East’s possible holdings of Kxxxxx

7.4.3 Redundancy

In the description of the interpreter algorithm in §7.3 we saw that as the possible play sequences identified by the planning stage are examined, \mathcal{C} -conjunctions are used to build up a picture of the circumstances under which particular branch selections will produce tricks. The procedure involved at MIN nodes in this algorithm involved the step of disjoining multiple \mathcal{C} -conjunctions. However, when two \mathcal{C} -conjunctions are combined in this way, the resulting expression usually requires some manipulation if redundancy is to be avoided. For example, consider the situation where we are trying to describe the possible worlds formed by three sequences of outstanding cards of length 2, 3, and 1, as shown in Figure 7.18 on the following page. Let us say that we are given the task of representing the disjunction of the possible worlds represented by

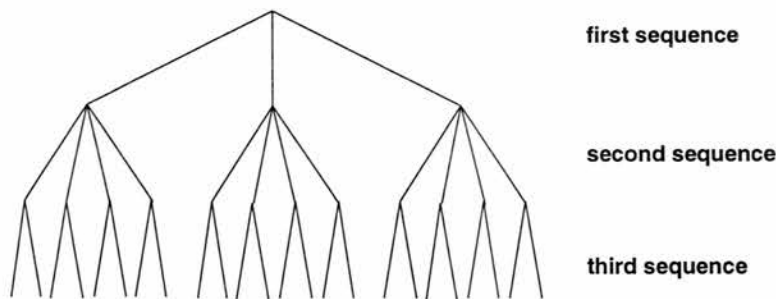


Figure 7.18: The composite tree of possible holdings for three outstanding sequences of lengths 2, 3 and 1

two separate \mathcal{C} -conjunctions depicted in Figure 7.19. Neither of these \mathcal{C} -conjunctions makes any restriction on the cards that can be held in the final sequence, but one \mathcal{C} -conjunction, represented in bold, allows East to hold *at least* one card from the first and second sequences, whereas the second, represented by dashed lines, allows *at most* one card.

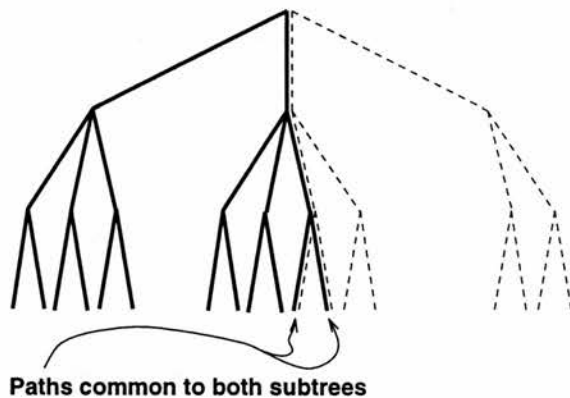


Figure 7.19: The paths allowed by the disjunction of two separate \mathcal{C} -conjunctions

If we assume that the final sequence in our pictorial representation is describing the low cards, and ignore for the moment the issue of the identity of the high cards, we can describe these two \mathcal{C} -conjunctions as

$$\begin{aligned} c_1 &: \text{east}(110) \wedge \text{east}(1110) \wedge \text{east_low}(11) \\ c_2 &: \text{east}(011) \wedge \text{east}(0011) \wedge \text{east_low}(11) \end{aligned}$$

The disjunction of these two \mathcal{C} -conjunctions could be represented by simply construct-

ing a term such as $c_1 \vee c_2$. However, it should be apparent that such an approach introduces the possibility of redundancy, as some paths may be represented twice by this expression. Indeed, in this example, there are two paths which are common to both the \mathcal{C} -conjunctions. In early versions of FINESSE, the language used to represent uncertainty did not readily allow the rewriting of such disjunctive expressions, leading to large expressions, sluggish performance and results which were non-trivial to interpret. This redundancy also complicated the task of generating probabilities or textual explanations from expressions — a feature which we describe later (*e.g.*, see Appendix B). FINESSE was therefore modified to manipulate any disjunction of \mathcal{C} -conjunctions into a form where every \mathcal{C} -conjunction in the disjunction refers to a completely disjoint set of paths. This is achieved by a process which we describe here in terms of a recursive algorithm making a top-down pass over the trees represented by two \mathcal{C} -conjunctions.

An Algorithm for Removing Redundancy

Given two \mathcal{C} -conjunctions to be disjoined together, we proceed as follows. Initially, the possible paths at the root of the trees are compared. If there are no branches in common, then the two trees are already disjoint and no modifications will be necessary. However, if there are branches in common, any of the paths containing these branches may be contained in both the trees, and a reformulation may therefore be necessary. The first step in this reformulation is to identify and separate the subtrees with no branches in common. These will be the subtrees which are rooted on branches which only occur in one of the original trees. In general there may be 0, 1, or 2 of these subtrees, depending on whether the trees have all their branches in common at the root, just one tree has some unique branches, or they both have unique branches. For the trees of Figure 7.19, there are two subtrees with no branches in common (see Figure 7.20 on the following page).

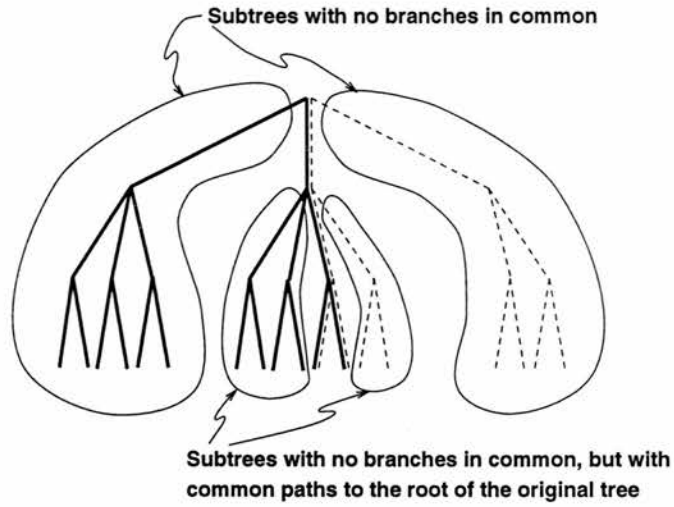


Figure 7.20: Steps involved in transforming overlapping subtrees into new subtrees with no overlap

Once these subtrees have been identified, the process is repeated on the tree found by descending one of the mutual branches (notice that any of the mutual paths may be chosen at this point, since the branching pattern at each node on any given level of the tree is the same). The subtrees produced when applying the modifications to this new tree have the paths from the root added to them to preserve the original structure. The two original trees in Figure 7.20, then, are transformed into a total of five new trees by this operation. In terms of the string representation described earlier, this process can be described as the algorithm of Figure 7.21 on the next page. This would convert the disjunction of the original c_1 and c_2 into the expression of (7.9), which has five disjuncts, each corresponding to one of the newly identified subtrees (here, in left-to-right order):

$$\begin{aligned}
 & \text{east}(100) \wedge \text{east}(1110) \wedge \text{east_low}(11) \vee \\
 & \text{east}(010) \wedge \text{east}(1100) \wedge \text{east_low}(11) \vee \\
 & \text{east}(010) \wedge \text{east}(0010) \wedge \text{east_low}(11) \vee \\
 & \text{east}(010) \wedge \text{east}(0001) \wedge \text{east_low}(11) \vee \\
 & \text{east}(001) \wedge \text{east}(0011) \wedge \text{east_low}(11)
 \end{aligned} \tag{7.9}$$

To rewrite the term $c_1 \vee c_2$ describing the disjunction of the \mathcal{C} -conjunctions c_1 and c_2 , call the following algorithm and disjoin all the \mathcal{C} -conjunctions it returns. If the algorithm fails, the \mathcal{C} -conjunctions already describe disjoint distributions and no rewriting is necessary.

Makes use of the two operators:

a AND b bitwise ‘and’ operation on the binary strings a, b .

a XOR b bitwise ‘exclusive or’ operation on the binary strings a, b .

1. Split c_1 into r_1 , the binary string representing the first outstanding sequence, and c'_1 , a term containing the remaining strings. Similarly split c_2 into r_2 and c'_2 .
2. Let $m = r_1$ AND r_2 .
3. If m is zero then the call to the algorithm fails (since the trees can have no paths in common).
4. Let $r'_1 = r_1$ XOR m (so that r'_1 identifies bits that are ‘on’ for r_1 but not for r_2). If r'_1 is not zero, combine r'_1 with the strings in c'_1 to form a new \mathcal{C} -conjunction and return this as a constituent of the final answer.
5. Let $r'_2 = r_2$ XOR m (so that r'_2 identifies bits that are ‘on’ for r_2 but not for r_1). If r'_2 is not zero, combine the r'_2 with the strings in c'_2 to form a new \mathcal{C} -conjunction and return this as a constituent of the final answer.
6. If m is not zero then call the algorithm on c'_1 and c'_2 , and combine m with the results of this call before returning them as constituents of the final answer.

Figure 7.21: Algorithm for removing redundancy in disjoint \mathcal{C} -conjunctions

7.4.4 Subsumption

Recall from our introduction of the interpreter algorithm that subsumption is necessary when a possible distribution is described by more than one C_j in a payoff profile

$$[T_1 \triangleright C_1, T_2 \triangleright C_2, T_3 \triangleright C_3, \dots] .$$

This is a similar situation to the redundancy we discussed above, where the constituent \mathcal{C} -conjunctions of a *single* term described overlapping distributions. Therefore, we model the process by a simple adaptation of the algorithm of Figure 7.21.

Given a profile in which there is no redundancy, FINESSE removes the first entry, $T_1 \triangleright C_1$, and uses it as the first entry in a new payoff profile. Each successive entry of the original profile is then inserted into this new profile, *after* processing it to remove distributions that are already described by C_j that are contained in the new profile. This process of removing overlap is carried out by using the algorithm of Figure 7.21 and only retaining the \mathcal{C} -conjunctions that are unique to the second term being compared. For example, consider again c_1 and c_2 from Figure 7.19 on Page 169. The tree representations of these \mathcal{C} -conjunctions have two paths in common, so the profile

$$[1 \triangleright c_1, 2 \triangleright c_2]$$

would require subsumption to remove the overlap. To do this, the pair $1 \triangleright c_1$ is inserted into a new profile list, followed by $2 \triangleright c'_2$, where c'_2 is the result of removing the paths in c_2 that are subsumed by c_1 . The value of c'_2 is found by applying the algorithm of Figure 7.21 and just keeping the paths unique to c_2 , which produces:

$$\begin{aligned} c'_2 = & \text{east}(010) \wedge \text{east}(0001) \wedge \text{east_low}(11) \vee \\ & \text{east}(001) \wedge \text{east}(0011) \wedge \text{east_low}(11) . \end{aligned}$$

The two disjuncts of this expression are the same as the final two \mathcal{C} -conjunctions of (7.9).

7.4.5 Tidying up

The transformation processes for subsumption and removing redundancy described above clearly increase the number of terms involved when disjoining \mathcal{C} -conjunctions together. For example, the two \mathcal{C} -conjunctions of Figure 7.19 were transformed into five after redundancy was removed. FINESSE therefore uses two techniques to simplify its results.

The first of these is to notice that in the final step of the recursion it is not necessary to separate out the branches which are unique to one of the trees. Since the trees being considered are only one level deep, it is trivial to transform two possibly overlapping trees into one by just forming the union of all their branches. The result of the final step of the recursion is therefore always just a single tree.

Secondly, a further rewriting procedure is used to shorten expressions by combining two \mathcal{C} -conjunctions which are slightly different into one. We call this a *collect* operation. Its operation is simply to check any disjunction of \mathcal{C} -conjunctions to identify any two whose strings differ in just one of their terms. Any such pair of \mathcal{C} -conjunctions can be re-written as a single new \mathcal{C} -conjunction which shares all the duplicated strings of the original, but replaces the one where they differ with the result of carrying out a bitwise ‘or’ operation on the two original strings. For example, in (7.9), the second and third \mathcal{C} -conjunctions differ only in the value of the middle string. Rewriting this produces a new \mathcal{C} -conjunction which again differs from the fourth \mathcal{C} -conjunction only in the value of the middle string. The original expression containing five \mathcal{C} -conjunctions can therefore be re-written into a disjunction of just three:

$$\begin{aligned} & \text{east}(100) \wedge \text{east}(1110) \wedge \text{east_low}(11) \vee \\ & \text{east}(010) \wedge \text{east}(1111) \wedge \text{east_low}(11) \vee \\ & \text{east}(001) \wedge \text{east}(0011) \wedge \text{east_low}(11). \end{aligned}$$

7.4.6 Generating Probabilities

Since the interpreter algorithm uses probability profiles to make its selections at MAX nodes, the procedure for actually calculating these profiles from the qualitative \mathcal{C} -

conjunctions is called upon many times. The overall performance of the interpreter algorithm is therefore very sensitive to the efficiency with which these probabilities can be generated.

In the earliest version of FINESSE [Frank 91], probabilities were produced by simply generating all the possible distributions of outstanding cards and then processing this set to eliminate those which violated the constraints imposed by the \mathcal{C} -conjunctions being considered. This approach is clearly cumbersome and also negates the main advantage of a qualitative representation language: that all the possible worlds need not be dealt with explicitly.

Subsequent versions of the system [Frank *et al* 92] used a more efficient approach based on the simplifying assumption that any outstanding card was equally likely to be held by either defender. Under this assumption, the probability associated with the basic form of \mathcal{C} -conjunction presented in (7.8) could be calculated using the formula

$$\prod_{i=1}^P \text{prob}(\text{Bin}_i), \quad (7.10)$$

where $\text{prob}(\text{Bin}_i)$ summed the Binomial distribution probabilities of each of the possible assignments of cards represented by the string Bin_i . Since the \mathcal{C} -conjunctions in the profiles produced by the interpreter algorithm always describe disjoint sets of possible distributions, the probability of any disjunction of \mathcal{C} -conjunctions was then found by applying the above analysis to each disjunct and summing the results.

However, the simplification of assuming that each card is equally likely to be with each defender introduces an error: in reality, the probabilities of individual cards being in each hand are not independent, since the outstanding cards must always be evenly distributed between East and West. This means that when calculating the probability of any given \mathcal{C} -conjunction, the contributions of each binary string, Bin_i , cannot be summed separately, due to the way in which assigning a number of cards to a defender's hand affects the number of cards subsequently necessary to complete it.

In general, there will be a total of N outstanding cards, of which East must hold N' (typically, $N = 2 \times N'$, unless we are considering a situation where one of the defenders

has already led to a trick). If East holds j cards from an outstanding sequence of length n_i , the number of ways in which his hand can be completed is then the number of ways of choosing $N' - j$ cards from the remaining $N - n_i$ cards. Under this constraint, the formula for the probability of East holding j out of n_i cards is therefore:

$$\frac{{}^{n_i}C_j {}^{N-n_i}C_{N'-j}}{{}^N C_{N'}}. \quad (7.11)$$

This constraint has the effect of making skewed distributions more unlikely, since if n_i is reasonably large, ${}^{n_i}C_j$ and ${}^{N-n_i}C_{N'-j}$ will both be far from their maximum values when j is small or close to n_i .

Counting the Possible Distributions

The current version of FINESSE uses an algorithm which observes the constraint of (7.11) by considering each `east` or `east_low` term in a \mathcal{C} -conjunction as being true in a particular subset of the set of total possible distributions between the two defenders. Finding the probability associated with a \mathcal{C} -conjunction is viewed as a problem in identifying the intersection of the subsets in which all of the constituent terms are true. The order of this set can then be divided by the total number of possible distributions to produce a probability.

This process has similarities to the situation in *incidence calculus*, where propositions are associated with *incidences* of possible world states in which they are true, rather than with probabilities [Bundy 92]. However, we are in the fortunate position of being able to generate (exhaustively, if necessary) the possible worlds in which any term of a \mathcal{C} -conjunction may be true, and hence do not at this stage need to make use of the facility offered by incidence calculus of deriving upper and lower bounds on the probability that an expression is true. In fact, the form of the expressions produced by FINESSE — a disjunction of \mathcal{C} -conjunctions — is particularly convenient, since the distributions represented by each individual \mathcal{C} -conjunction are disjoint. This means that producing the *actual* distributions represented by each \mathcal{C} -conjunction in a disjunction is optional, since the distributions described by each disjointed \mathcal{C} -conjunction will have empty intersections. We make use of this feature by simply counting the *number* of

distributions allowed by an individual \mathcal{C} -conjunction. The probability of a disjunction of \mathcal{C} -conjunctions can then be found by summing the number of distributions allowed by each individual \mathcal{C} -conjunction and then dividing by the total number of possibilities.

Example

Let us first consider what is involved in calculating the probability of a \mathcal{C} -conjunction by manually carrying out this process for a simple example. For this purpose, we will return to the \mathcal{C} -conjunction which we introduced in §7.4.2, and depicted in Figure 7.17 on Page 168:

$$\text{east}(11, [\text{king}]) \wedge \text{east_low}(001010). \tag{7.12}$$

Recall that this \mathcal{C} -conjunction describes the possible distributions of Kxxxxx where either East or West hold the King and East holds one or three low cards.

In order to calculate the probability of these distributions we begin by calculating the chances of the possible splits of the six outstanding cards between the two players (*e.g.*, 3–3, 6–0). Assuming that we are considering normal Bridge, we can find these probabilities by substituting the values $N = 26$, $N' = 13$ and $n_i = 6$, into (7.11) for the possible different values of $j = 0, \dots, 6$. These probabilities are summarised in Figure 7.22.

Split	Probability	Number of cases
0–6	0.007	1
1–5	0.073	6
2–4	0.242	15
3–3	0.355	20
4–2	0.242	15
5–1	0.073	6
6–0	0.007	1

Figure 7.22: The probabilities of the possible splits of 6 cards

Now let us consider *how many* distributions are actually described by the \mathcal{C} -conjunction

of (7.12). This is done using simple combinatorics, as shown in Figure 7.23. Combining

Split	East's holding	Number of cases
1-5	x	${}^5C_1 = 5$
2-4	Kx	${}^5C_1 = 5$
3-3	xxx	${}^5C_3 = 10$
4-2	Kxxx	${}^5C_3 = 10$

Figure 7.23: The number of distributions described by (7.12)

these two tables together, we can produce a probability for the truth of (7.12), as shown in Figure 7.24. Thus, we produce the answer 0.481. Comparing this to the

Split	Probability	Total Cases	Cases in (7.12)	Total contribution
0-6	0.007	1	-	0
1-5	0.073	6	5	$(5/6) \times 0.073 = 0.061$
2-4	0.242	15	5	$(5/15) \times 0.242 = 0.081$
3-3	0.355	20	10	$(10/20) \times 0.355 = 0.178$
4-2	0.242	15	10	$(10/15) \times 0.242 = 0.161$
5-1	0.073	6	-	0
6-0	0.007	1	-	0
			Total	0.481

Figure 7.24: Calculating the probability of the distributions described by (7.12)

result which would be generated by (7.10), we see that this equation would calculate the value $prob(11) \times prob(001010)$. The first of these probabilities evaluates to one, as it places no restrictions on the outstanding cards. The second is given by:

$$prob(001010) = {}^5C_1 \left(\frac{1}{2}\right)^5 + {}^5C_3 \left(\frac{1}{2}\right)^5 = \frac{15}{32} \simeq 0.469.$$

Thus, ignoring the constraint that the defenders must equally share the outstanding cards leads to incorrect results.

The Shape-list Algorithm

In order to automate the probability generation we carried out manually above we use an algorithm which processes any given \mathcal{C} -conjunction to produce figures equivalent to the third column of Figure 7.24. To do this, we arbitrarily choose East's perspective (the situation is symmetrical) and construct a list of 14 numbers, in which the n th number denotes how many distributions East may hold that contain $n-1$ cards. We refer to this structure as a *shape-list*. For any \mathcal{C} -conjunction, the corresponding shape-list, *list_out*, can be found using the algorithm of Figure 7.25.

-
1. Set *list_in* to [1,0,0,0,0,0,0,0,0,0,0,0,0,0].
 2. Set *list_out* to [0,0,0,0,0,0,0,0,0,0,0,0,0,0].
 3. Select an unexamined term from the \mathcal{C} -conjunction, with the string *Bin_i*. Let n_i be one less than the length of *Bin_i*.
 4. For all $0 \leq j \leq n_i$:

If the j th bit in *Bin_i* is 1 then shift *list_in* to the right j places by inserting j zeros, multiply each entry by ${}^L C_j$ and add the result to *list_out*.
 5. If there are unexamined terms remaining in the \mathcal{C} -conjunction, set *list_in* to equal *list_out*, set *list_out* to [0,0,0,0,0,0,0,0,0,0,0,0,0,0], and goto Step 3.

Figure 7.25: Algorithm to produce a shape-list for a single \mathcal{C} -conjunction

Let us examine the operation of this algorithm by considering how it behaves on our example \mathcal{C} -conjunction. In Figure 7.26 on the following page we show how the values of the variables used by the algorithm change during its execution. Entries are made in the table whenever the value of a variable is set or updated.

First, *list_in* and *list_out* are initialised. Initially, before any terms have been considered, the only possibility in *list_in* is that East has no cards. *list_out* is initially composed entirely of zeros.

At Step 3 we assume that the string 001010 is selected (the order of selection is unimportant). This string has two bits set to one, representing the possibilities of adding either 1 or 3 low cards to East's hand. To add just one card, *list_in* is shifted one place to the right, but this low card may be any one of the 5 available, so the resulting

Step	list_in	list_out	String	i
1	[1,0,0,0,0,0,0,0,0,0,0,0,0]	[0,0,0,0,0,0,0,0,0,0,0,0,0]	001010	1 3
2				
3				
4				
5	[0,5,0,10,0,0,0,0,0,0,0,0,0]	[0,0,0,0,0,0,0,0,0,0,0,0,0]	11	0 1
3				
4				

Figure 7.26: Trace of the shape-list algorithm

list is also multiplied by 5. Similarly, adding three low cards is achieved by shifting `list_in` to the right by three places and multiplying by ${}^5C_3 = 10$. Thus, the lists

[0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0]

are added to `list_out` by a process of list addition, in which entries in corresponding positions in the two lists are summed to produce the entries of the new list. Thus, we have so far assigned either one or three cards to East’s hand, and when we next arrive at Step 3, the value of `list_in` reflects this (there are five possible distributions in which East may hold 1 card and ten in which he holds 3). Step 4 then recalculates `list_out` to take account of the cards represented by the binary string 11. Thus, the way in which a single string can represent many distributions is accounted for by Step 4, and the conjoining together of `east/2` and `east_low/1` predicates into a \mathcal{C} -conjunction is accounted for by making the `list_out` of each successive loop the `list_in` of the next.

As we can see, the values of the final `list_out` correspond to the required numbers in Figure 7.24. To produce a probability from a shape-list, we then simply carry out a procedure similar to that above. For the generic \mathcal{C} -conjunction of 7.8, the shape-list will have been produced by considering

$$n = \sum_{i=1}^p n_i$$

cards. If there is again a total of N outstanding cards, of which East must hold N' ,

and the function *ith_entry* returns the *i*th entry of a shape-list, the probability that the distributions represented by a given shape-list, *List*, will actually occur is given by the following modified form of (7.11):

$$\sum_{i=0}^{N'} \frac{ith_entry(i, List) \times {}^{N-n}C_{N'-i}}{{}^NC_{N'}}.$$

7.5 Coping with Non-locality in Bridge

Besides constraining the search space, it transpires that our use of tactics to construct the single-suit trees of possibilities in Bridge has another advantage; it allows us to specify plays which the defence should *never* consider in response to a particular play by the declarer. Let us return to our example illustrating non-locality from the previous chapter, repeated in Figure 7.27.

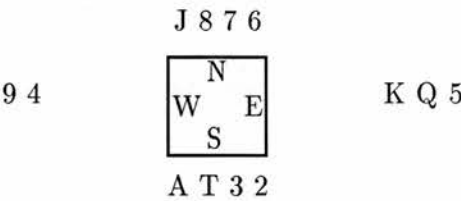


Figure 7.27: A Bridge example giving rise to non-locality, repeated

Recall that if the declarer does not know the actual distribution of the outstanding cards in this situation, his best play is to lead the 6 from the North hand and play low from the South hand unless East plays the 9 or one of the King or Queen (this play is actually a Type 2 finesse of the 6).

Faced with the finesse of the 6, East’s best option if he holds the cards shown is to play low with the 5. If he does this, West will win the trick with the 9 and the declarer will be restricted to just two tricks in the suit. Similarly, if East starts with Kx or Qx, the best option will again be to play low. We found, however, that if East plays the King and West plays the 4 the resulting situation (shown in Figure 7.28 on the following page) is mis-analysed because of non-locality.

In fact, it is possible to use the high-level representation of tactics to help us in this

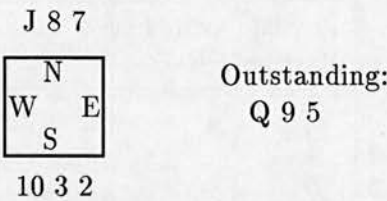


Figure 7.28: Second round situation (giving rise to non-locality)

situation. Specifically, when any finesse tactic is analysed by the interleaver, we can make the following inferences:

- If the second player plays higher than the finesse card, then
 - either* the finesse would have succeeded. That is, the second player holds *all* the cards from the outstanding sequence immediately above the finesse card,
 - or* the finesse would have failed, but the second player had no cards lower than the finesse card.

To see how the inferences made by these rules would affect our example above, consider what happens when the declarer tries to finesse the 6 in the initial situation. If East plays the King, there are two possible inferences. Firstly, we could infer that the finesse was going to succeed and that East therefore holds the 9 (the only card in the first outstanding sequence above the finesse card). Alternatively, we could infer that the finesse would have failed, and although West has the 9 East has no cards below the finesse card. Both these inferences can be easily represented in the uncertainty language of FINESSE and carried as a disjunction by the interpreter algorithm.

The significance of these inferences can be seen by recalling the table describing the possible worlds under which the finesse of the Jack and the finesse of the seven will succeed in the second round situation (see Figure 7.29 on the next page).

If we examine the worlds to see which meet the constraints that East holds the 9, or doesn't hold the 9 but has no cards lower than the 6, we find that we rule out just two: the ones where East holds the Q5 or the singleton 5. These are exactly the two

Possible worlds: East holds	finesse of the 7	finesse of the J
Q95	•	•
Q9	•	•
Q5		•
95	•	•
Q	•	•
9	•	•
5		•
void	•	

Figure 7.29: Possible worlds under which second round tactics produce 2 tricks

which we identified in the previous chapter as giving rise to non-locality. Omitting them results in the correct choice (the finesse of the 7) being made in this state.

With these heuristic inferences incorporated, *FINESSE* correctly identifies the finesse of the 6 as the best line of play in the original situation, with a probability of 0.4604 of making three tricks. This compares with the result of just the basic interpreter algorithm which selects a line of play involving a Type 1 finesse of the ten, followed by the cashing of the Ace, with a probability of success of 0.458.

The ability to make heuristic inferences about the finessing tactics significantly improved *FINESSE*'s performance, as demonstrated by the results of Appendix C. It is important to note that these inferences are made by considering the type of play being carried out, and not by analysing the tree of possibilities. Although they may appear similar in nature to the inferences made during the doublethink algorithm, they are only possible because of the high-level nature of the tactic representation.

7.6 Summary

We have examined possible techniques for coping with non-locality more efficiently than exhaustive strategy minimisation. We identified the benefits that could be gained from representing information qualitatively and by making inferences about the possible worlds under which each internal node of a game tree could be reached.

We applied some of these ideas to Bridge, describing in particular an algorithm that selects lines of play by using qualitative information to represent all the possible worlds efficiently, and a heuristic to cope with the effects of non-locality. We also showed how our representation language supported the generation of probabilities, and mentioned other benefits of describing information qualitatively, such as the ability to generate textual explanations (which is implemented in *FINESSE*) and the possibility of incorporating inferences drawn from the bidding.

When considering probability generation, we noted the flaw in assuming that each card was equally likely to be with either defender, and described an algorithm which, without resorting to the generation of all the possible distributions of the outstanding cards, respected the constraint that each player will usually hold half of the outstanding cards.

Chapter 8

Interleaving Plans With Dependencies

Wot — no quote?

— ALAN BUNDY

As we have seen, the overall FINESSE architecture uses the common problem-solving strategy of breaking a task down into a series of smaller subproblems, which are then solved independently. The single-suit lines of play that have emerged from our discussions in the previous three chapters are the first stage in this process, allowing the subproblems of individual suits to be tackled. However, it is widely recognised that although such subproblems are easier to solve than the original task, their independently derived solutions may not combine to produce a good overall solution. Some possible complications are:

- The individual plans may contain dependencies which make it difficult or impossible to execute them together.
- There may be domain actions that are overlooked because they result from considering more than one goal simultaneously.
- The best solution to the overall problem may not be composed of the best solutions to each of the subgoals, but instead of sub-optimal solutions to some or all of them.

The second and third of these issues are examined in Chapter 9 and Chapter 10,

respectively. In this chapter, we tackle the first: that of combining, or *interleaving*, together a number of independently derived plans that may have dependencies.

Since Bridge is a game with incomplete information, we know from Chapter 6 that the task of selecting a strategy is doubly exponential in the number of MAX levels in the game tree. However, the interleaving task offers a way around this complexity. For interleaving involves choosing an *order* for some *given* operators rather than actually choosing which operators to include. We will therefore introduce a tractability refinement which assumes that it is possible to specify constraints that remove *all* possible interactions between the plans being interleaved. If this is the case, we will be able to simply consider the chance of success of a *particular* ordering obeying the constraints, rather than all of them. This allows us to avoid non-locality because when constructing this ordering it will be possible to consider as equal any choices at points where there are multiple operators obeying the constraints. We therefore never actually choose between operators with possibly different results, which is what could give rise to non-locality. Of course, the cost of constructing even a single ordering is still exponential in the length of the game, but this is an improvement over doubly exponential complexity.

In fact, constructing a single ordering allows us to provide an informal justification for the overall approach of tackling card play by breaking it down into single-suit subproblems. Both the original problem and these subproblems have doubly exponential complexity in the number of MAX levels. However, the trees in the single-suit problems typically involve many fewer levels than those of the complete game. Thus, the approach of producing and then combining together solutions for single-suit problems reduces the value of the exponent in the doubly exponential stage and subsequently incurs an exponential, rather than doubly exponential, cost when interleaving.

8.1 The Problem

We should make it clear that, in this context, the word ‘interleaving’ does not refer to any attempt at combining planning with execution, but describes the process of establishing, pre-execution, the executability of a set of plans. When playing Bridge,

this ability is clearly important since (unless the opponents are *exceedingly* generous) the option of backtracking when actions fail to work out as expected is not available.

The main reason that interleaving turns out to be difficult (and interesting) is that the number of possible interleavings allowed by even a relatively simple set of plans precludes exhaustive analysis. Below, we will explore this, and set out the basis for a solution that involves explicit reasoning about particular dependencies that plans may contain.

8.1.1 Combinatorial Explosion

The single-suit lines of play produced by FINESSE are trees, with the branches at any node either corresponding to declarer tactics or to possible plays by the defenders. These plans are ‘deterministic’ in the sense that, at each point where the declarer controls the choice of which branch to follow, one (or more) options are indicated as being superior to all the others. When trying to interleave such lines of play, we come up against a number of potential sources of combinatorial explosion.

Ordering the Tactics

Within an individual line of play, any two tactics that could possibly be executed in the same play of a game are strictly ordered. However, in trying to find an optimal interleaving we could find ourselves having to consider all the possible ways of forming legal play sequences for the entire game.

Even if we assume that the lines of play are linear (that is, there is only one possible response by the defence to each tactic), the number of possible orderings quickly becomes very large. For example, for the simplified lines of play shown in Figure 8.1 (where each MAX branch represents a tactic to which there is only *one* MIN response, and the plans contain n_1 , n_2 , n_3 , n_4 tactics respectively) the number of possible interleavings is given by the formula:

$$\frac{(n_1 + n_2 + n_3 + n_4)!}{n_1!n_2!n_3!n_4!}.$$

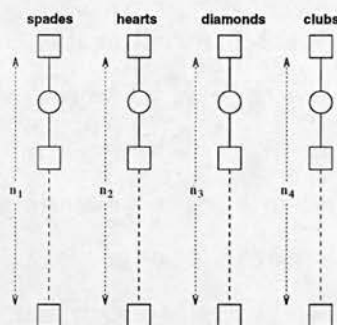


Figure 8.1: Linear lines of play

For typical lines of play, this number quickly becomes very large. For example, if $n_1 = n_2 = n_3 = 4$, and $n_4 = 1$ there will be 450450 possible interleavings.

Disjunction Caused by Uncertainty

Of course, the representation of Figure 8.1 is over-simplistic, as it ignores the different possible responses open to the defenders when any single tactic is executed. This disjunction (the defenders may only choose *one* of the available alternatives) makes the task of examining the possible interleavings still harder, as the uncertainty over the defenders' choice of cards means that we can now no longer be sure of the situation reached after the execution of a tactic. This not only adds extra branching to the process of looking at the possible orderings, but also complicates the task of deciding whether preconditions are actually true at a given point in a plan, since many actions will only be possible in a subset of the possible worlds (for example, East can only play the $A\spadesuit$ if he actually holds the $A\spadesuit$).

Coping with the Opposition

Finally, the above ignores the possibility that we may not always have complete control over the ordering of actions. In Bridge, for example, the defenders may be able to spoil declarer's plans by guessing and then disrupting the order in which he wishes to carry out particular actions. Ensuring that the defence are not given such opportunities can be an important part of card play.

This complicates matters since it forces us to consider, after any tactic which may surrender the lead, every possible continuation that the defence could then choose. This again adds to the search involved when forming a global plan, by introducing disjunctive branching over which we have no control. In fact, the interleaving task in the domain of Bridge can be viewed as a giant incomplete information search problem; the tree of possibilities has MAX branches for each tactic that MAX can select from the lines of play when it is his turn to lead, and MIN branches both for the possible responses to these tactics and for the possible tactics that MIN may select when it is the defence's turn to lead. The presence of incomplete information, of course, introduces the complications of non-locality discussed in Chapter 6 and Chapter 7.

8.1.2 Reasoning about Dependencies: Resources

The most well-known example of conflict in independently derived sub-plans is the Sussman anomaly, which we discussed in §3.2. In this problem, solving one subgoal undoes the achievements of the plan for solving the other. It is therefore *necessary* to interleave the plans for these goals, and also to constrain this interleaving to ensure that any operator which may undo a goal condition occurs before the point where that goal is established.

As we noted above, however, the disjunction caused by the incomplete information in Bridge makes it difficult to decide whether any condition holds at a given point in a set of plans. Further, existing systems for planning under uncertainty operate by introducing separate branches for each possible world (or contingency) created by the outcomes of the operators with uncertain effects. The number of possible worlds in Bridge, however, clearly makes this approach prohibitively expensive. We therefore develop an alternative architecture which, without explicitly considering all the orderings of the operators in a set of plans, explores the possible interleavings without having to examine each contingency separately. This architecture will function by firstly identifying any dependencies which may cause conflicts, and then reasoning about how the plans will have to be constrained in order for these dependencies not to cause a problem. Some frameworks within which such constraints can be incorporated will be introduced later in the chapter. First, however, we will examine how the possible dependencies

may be formalised.

Domain Clichés

In Chapter 3 we introduced Chapman's intractability theorem for the truth criterion of domain-independent planners involving extended representations (such as disjunction). This clearly indicates that the task of constructing a completely general planner capable of working in the Bridge domain is likely to be a vain pursuit. However, it does not necessarily imply the converse — that a planner capable of working in the Bridge domain must be completely dependent on domain-specific features. Chapman himself, for example, suggests three ways in which the problem of an NP-hard truth criterion for general planners might be overcome. Loosely, the first of these is simply to 'hope for the best' and trust that the problems of combinatorial explosion do not prove too serious. However, we have already seen that the combinatorial problems in Bridge play are formidable. Further, Chapman's second suggestion — to relax the correctness requirement, producing a heuristic planner whose plans would sometimes not quite work — also seems ill-suited to a domain in which backtracking is not an option when plans 'go wrong'. In discussing a third alternative, however, Chapman notes that it is often easy to find an efficient truth criterion for specific domains. Although he concedes that these criteria are often quite different, he conjectures that it may be possible to build a system based on *intermediate techniques*, which are neither completely general, nor completely domain-specific. He says:

I envision a cliché-based constraint-posting planner for extended action representations which would have truth criteria specific to *clichés* that operators in the world might instantiate. A planner with a few dozen clichés might well cover most interesting domains.

In this context, *clichés* are formal structures occurring in the domain. As an example of a cliché, Chapman discusses *resources*:

An instance of a *resource cliché* consists of a state variable in the world which holds a quantity in some total order, together with at least one *consumer operator*, which decreases, relative to the order, the value of the state variable, and at least one *dependent operator* which has as a precondition that the state variable have a value greater than some threshold. There may also be *producer operators* that increase the value of the state operator.

Below, we will develop this concept of a resource-based cliché, and show in particular how the dependencies between single-suit lines of play in Bridge can be described in terms of the consumption of a type of resource known as *leads*.

8.1.3 Resources in Bridge: Leads and Entries

One of the important features of Bridge declarer play is that the order in which the players lay their cards can have a decisive effect on the outcome of a trick. Indeed, we have already encountered manoeuvres like the finesse which will only succeed if played in a certain way. Manipulating the *lead* (the player who must start the next trick) to be in the right place at the right time is a key skill for all Bridge players. Ensuring that it is possible to switch between hands at the necessary moments is known as establishing *communication*.

A human declarer will typically try to establish good lines of communication between his hand and dummy's hand, and may also attempt to disrupt the communication between the defenders' hands. FINESSE's single-suit planning algorithm, however, ignores this issue completely, so it is almost inevitable that the individual lines of play will not be executable unless tactics from other suits are available at critical moments to manipulate the lead.

Leads

It is possible to treat the position of the lead as an example of a resource cliché, with FINESSE's tactics playing the role of consumer and producer operators (since in general their execution will require the lead to be in one of the North or South hands, and will leave the lead with one of the four players). For example, cashing the Ace in the situation of Figure 8.2 on the following page would require the lead to be in the North hand and would leave the lead in the North hand after execution (assuming a No Trumps situation).

The demands made by the individual tactics on this type of communication requirement are relatively easy to specify. In FINESSE, there are only three types of tactics that



Figure 8.2: A simple cashing example

require the lead to be in a specific position in order to be executable. They are:

- **Cash** tactics, which require the lead to be manoeuvred into the hand containing winners, if the opposite hand is void (as above).
- **Sequence** tactics, which also require the lead to be manoeuvred if the hand opposite the sequence is void.
- **Finesse** tactics, which require the defender being finessed to be the second player in the trick.

However, these requirements on the positions of the leads are at the level of individual tactics, whereas the task we will be considering in this chapter is the interleaving of complete lines of play. When describing the resource requirements of such composite plans, we will use a slightly more advanced terminology.

Entries

Consider the situation of Figure 8.3 on the next page. Nine tricks can be won here with a plan that involves playing sequence tactics until the defenders part with their Ace, and then cashing the remaining winners. Each of these tactics requires the lead to be in the North hand, so at first glance it may seem that the appropriate communication requirements would be for the lead to manoeuvred into the North hand ten times. However, this of course ignores the fact that each sequence and cash tactic (except the sequence tactic that loses to the Ace) will also leave the lead in the North hand after execution. The lead will therefore be in the correct place for each successive tactic and

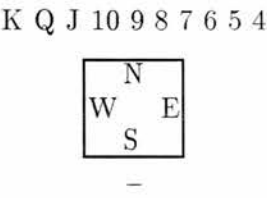


Figure 8.3: An almost solid suit

communication will only need to be established twice (once to initiate the plan, and a second time to re-enter the North hand once the defenders have cashed their top Ace). Human Bridge players tend to take these higher level considerations into account when considering card combinations, with the result that they more commonly refer to the *entry* requirements of lines of play than the positions of the lead required by individual tactics. Thus, the example above would be described as requiring two entries into the North hand in order to be tackled successfully.

We too will use entries as a resource cliché when describing the resource requirements of lines of play. However, notice that the number of entries required by a line of play may vary depending on the distribution of the outstanding cards, or even how the defenders choose to play their cards. We will discuss this in more detail in §8.2.2. For now let us just say that, given a line of play, and a particular set of responses by the defenders, an entry is required for every tactic:

- that requires the lead to come from a specific hand, as described above, and
- for which the previous tactic in the line of play leaves the lead in some hand other than that required.

Since the tactics within individual lines of play are strictly ordered with respect to each other, such entry requirements would have to come from one of the other suits.

8.1.4 Coping with an Opposition

An algorithm based on an analysis of the entry requirements described above would be capable of solving the kind of problem shown in Figure 8.4 on the following page

where the contract is 6 No Trumps.

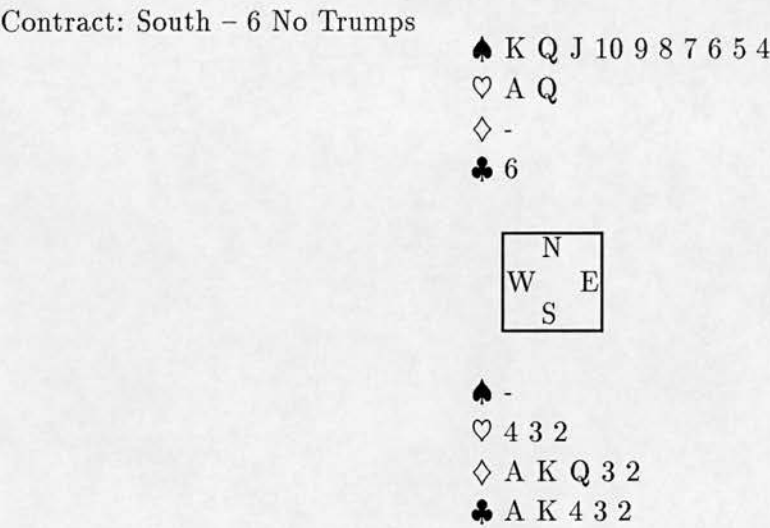
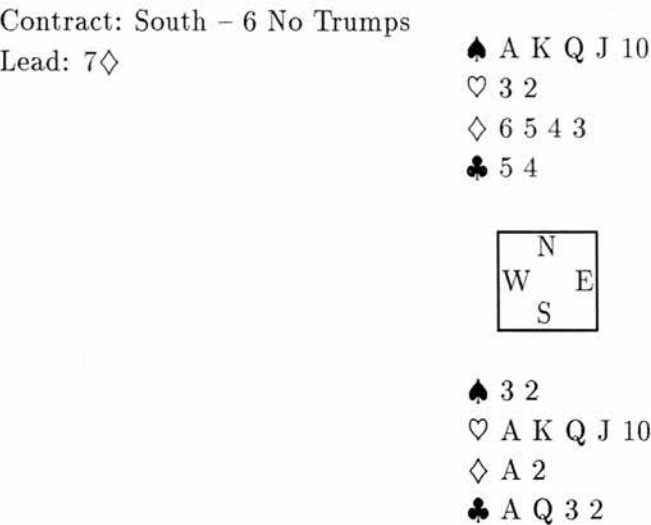


Figure 8.4: A possible deal containing an almost solid suit

Against any lead, the declarer can only hope to make the contract if he can establish the spade suit, and as we have already seen this requires two entries into the North hand. A similar analysis of the entries produced by the heart suit would reveal that the lead will be left in the North hand twice under the condition that West holds the $K♥$. Therefore, there is a global plan which will lead to 12 tricks whenever the $K♥$ is with West, but only six tricks otherwise.

One problem that would not be foreseen by such resource reasoning, however, is that even when it is possible to construct a plan capable of returning the required number of tricks, the plan may not be executable if the defenders can first take enough tricks to break the contract. Under these circumstances, declarer would be forced to throw away cards that were earmarked as winners in his original plan. For example, playing to make 6 No Trumps against a diamond lead in the situation of Figure 8.5, declarer might get carried away and rather than settling for cashing his top tricks to make the contract, he may go for an overtrick with the club finesse. Unfortunately, if this finesse was to fail the contract would almost certainly be defeated, as the defender winning with the $K♣$ would surely continue with another diamond if he held one. Preventing this from happening in an automated planner calls for the ability to reason about the options open to the defenders if they are given the lead.



8.2 Resource Profiles

Reasoning about the resource analyses of lines of play first requires some way of identifying what these analyses are. This section presents a discussion of how such information can be produced.

8.2.1 Context-dependency

We have seen already that the resources consumed by tactics may depend on the actual world pertaining when the tactic is applied. In Bridge, this dependency is determined by the cards played by the defenders, and therefore has the consequence that any resource production and consumption is likely to be context-dependent. For example, a finesse tactic in the simple card combination where North-South hold the $AQ\spadesuit$ opposite the $2\spadesuit$ would require an external entry into the South hand, and leave the lead in either East's or North's hand depending on the position of the King.

For the same reasons that FINESSE deduces the effects of its tactics according to the context in which they are applied rather than attempting to exhaustively specify all the possible postconditions, it would be impractical to attempt to incorporate resource information into the actual representation of tactics.

Instead, we incorporate information on the position of the lead into the game state (it was not necessary to include this until now, since the single-suit planning algorithm ignores this detail) and, when selecting lines of play, determine not only how many tricks can be produced from them but also the corresponding resource analysis.

Bottom-up Analysis

We have already seen that two entries are required in order to establish and win nine tricks in the situation of Figure 8.3, repeated in Figure 8.6 for ease of reference. Given that the resource analysis of a line of play may differ for different sequences of moves by the defenders, let us initially examine just the case where the Ace is played by West on the first round of the suit, and move on to the more general situation later. Figure 8.7 on Page 198 shows a rendering of the line of play for this situation which represents the

K Q J 10 9 8 7 6 5 4



Figure 8.6: An almost solid suit, repeated

resources consumed and produced by each individual tactic with `lead/1` predicates.

Summing up all the entries in the ‘consume’ and ‘produce’ columns would tell us that ten entries into the North hand are required, and nine leads in the North hand are generated, so by simply ‘matching’ the production to the consumption we could conclude that one entry into the North hand would be required by this line of play. Unfortunately, as we have seen, this conclusion is wrong. In fact, the suit requires two entries to establish; one to play the sequence tactic, and another to subsequently re-enter the North hand to cash the established winners. The flaw in simply matching up the consumption and production is that the quantities being treated as resources are generated and consumed over time, whilst the simple idea of summing their values does not respect this dependency. In our example, for instance, the final lead produced cannot be used to supply the initial entry requirement. However, this constraint is easily accommodated when we incorporate the analysis of resources into the interpreter algorithm from the previous chapter. This algorithm makes a bottom-up pass of the tree, so the only effects it is possible to see when examining any node are those created by tactics which have already been applied. This allows us to safely determine whether the conditions on entries we specified in §8.1.3 are present. For complete lines of play, we will represent entry consumption by the term `entry(Hand,N)`, where `Hand` is the hand into which entries are required and `N` is the total quantity. Figure 8.8 on Page 199 shows how a resource analysis using such terms can be built up. Notice that in effect resource supply is ‘cancelled’ with resource production as we move up the tree, with any unused supply being left over and available to supply tactics in lines of play from other suits (or resulting in leads by the defence).

The resource analysis for the plan is constructed from the bottom up, so the first tactic

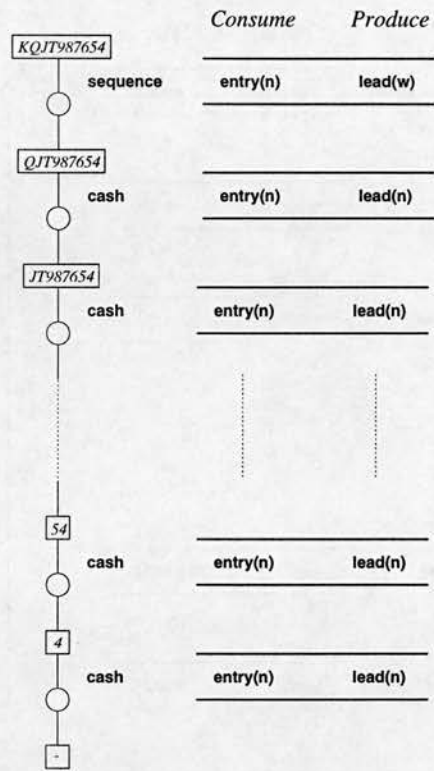


Figure 8.7: Resource consumption and production of tactics in a single line of play

to be considered is the final cash tactic, which in isolation would both require and leave the lead in the North hand. However, the tactic immediately prior to this also leaves the lead in the North hand, so the conditions for requiring an entry are not present, and the resource analysis is therefore just that the lead will be left in the North hand once. This cancelling effect continues up the tree until the first cash tactic is reached. This tactic does require an entry, since the previous trick was won by West's Ace. Similarly, the initial sequence tactic has an un-supplied entry requirement since it is not possible to make assumptions about where the lead will be on the first trick. The number of entries required in the final analysis therefore goes up to two, with the lead being left in both the North hand and the West hand once.

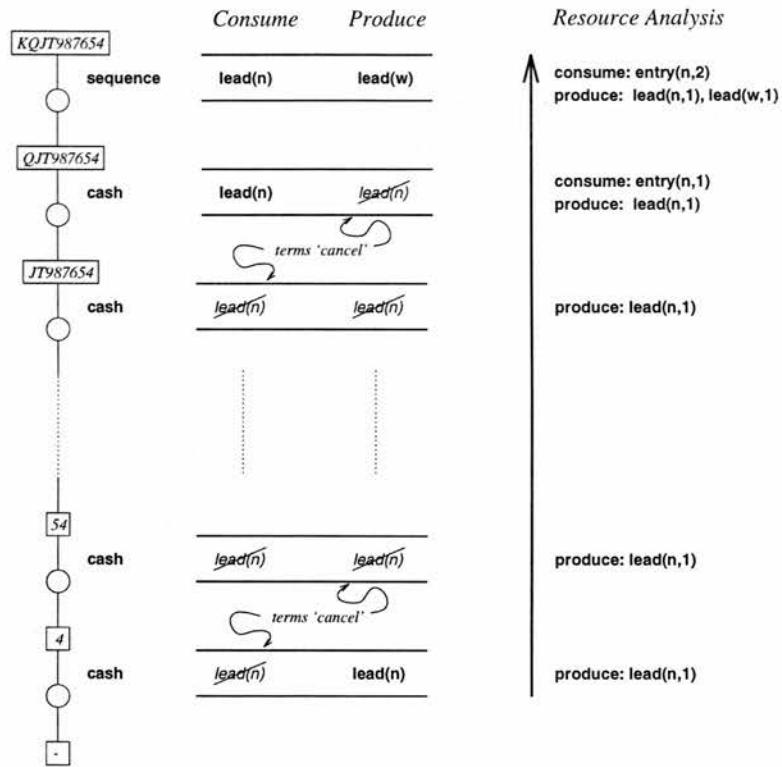


Figure 8.8: Producing the resource analysis of a line of play bottom-up

8.2.2 Considering all Possible Worlds

The above example is obviously a simplification, since it assumes that the outstanding cards will be played in just one particular way. In general, a line of play will have MIN nodes with more than one branch, so we will have to cope with uncertainty over the opponents' choices. Fortunately, though, as well as naturally respecting the temporal nature of resource consumption and production, the bottom-up nature of the process outlined above also enables us to incorporate the resource analysis step into the interpreter algorithm described in the previous chapter.

Recall that at any MIN node, the defenders can restrict MAX to the minimum possible payoff in each world. To model this, we have seen how the interpreter algorithm uses the notion of subsumption: first the basic payoff profile is generated, and then, starting with the smallest payoff and gradually working up, any overlap is gradually removed from the remaining terms.

To produce a profile based on resources, rather than the numbers of tricks we initially used in our definition of a payoff profile, we need to update our notion of ‘payoff’ so that it takes into account the possibility of differences in resource production and consumption.

Payoff -tuples

In FINESSE, there are in fact seven different possible resources that can be consumed or produced, as summarised in Figure 8.9.



Figure 8.9: Possible resources that can be (a) Required, and (b) Produced

With the exception of the `lead(ew)` term, the meanings all of these terms should be clear from our earlier discussion of resources in §8.1.3. The additional `ew` possibility is included to cope with a possible consequence of the duck tactic.

Recall that ducking involves the declarer playing a low card from both his hand and from dummy in an attempt to improve his chances in a suit. An example situation where the declarer could well consider ducking is shown in Figure 8.10. There are five cards outstanding in this suit, so if the Ace and King are cashed immediately at least one defender will still have cards remaining. Without any entries supplied by other suits, then, the declarer would be unable to win with any of the low cards in the North hand. To improve his chances, declarer could instead duck one round in the suit before cashing the masters. This will allow three of the low cards in the North hand to win tricks whenever the outstanding cards split 3-2 between the defenders (under our assumption that there are no trumps).

Among the MIN branches of a duck tactic applied in this situation will be one which represents the case where both defenders play low cards (‘x’s). Since the declarer also

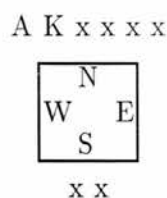


Figure 8.10: A ducking example

ducks by playing his two lowest cards, all four cards played on this trick will appear to have the same value. This makes it difficult to decide who has won the trick.

FINESSE resolves this problem by assuming firstly that the defenders will not be kind enough to let the declarer win a trick with an x card. (This is justified, since FINESSE only ever ducks with cards that can be beaten by outstanding x cards). The winner, then, must be one of the defenders. However, there is no way of knowing which. Since there is no way of expressing this disjunction within FINESSE's uncertainty representation language, an extra type of resource describing a 'lead in either the East or West hand' is introduced.

FINESSE therefore re-defines payoffs as follows:

Definition 8.1 *A line of play is said to have a payoff $\pi: \langle T, E_n, E_s, L_n, L_s, L_e, L_w, L_{ew} \rangle$, under some possible world, if against best defence it leads to T tricks, requires E_n entries into the North hand, E_s entries into the South hand, generates L_n, L_s, L_e, L_w leads in the North, South, East, and West hands, and an additional L_{ew} leads in either East's or West's hand.*

Augmenting the Interpreter Algorithm

To incorporate the above change into the interpreter algorithm, the following modifications are necessary:

- Partition the set of possible worlds at each node according to the payoff 8-tuple that describes the line of play beneath that node, rather than just the number of tricks generated by the line of play.

- Define a binary relation on the new payoff 8-tuple that can be used to determine which of any two given profiles is the ‘worse’.
- Modify the subsumption process so that it starts with the lowest payoff (according to the new relation), maximises the number of worlds assigned to it, and then repeats this for the other payoffs, in ascending order.

There are clearly many ways in which the payoff of a single-suit Bridge problem can be assessed. As well as the obvious goal of restricting the number of tricks that the declarer can take, the defenders may be interested in limiting his ability to generate entries into either or both of his hands, or in gaining as many tricks as possible themselves. The declarer in turn may be interested just in the number of tricks or may actually be willing to give up tricks in one suit in order to generate entries that can be used to reach more promising suits. We will return to this issue in Chapter 10, when we discuss the overall system architecture in more detail. For now, however, we will assume as a default that the players’ behaviour is described by the relation given below.

A Relation Between Payoffs

For any given payoff, we will be interested in three features: the number of tricks won by the declarer, the total number of entries required to establish these tricks, and the total number of leads given to the defence in the course of achieving this. Given a payoff, $\pi = \langle T, E_n, E_s, L_n, L_s, L_e, L_w, L_{ew} \rangle$, let us represent these quantities by $tricks(\pi) = T$, $entries(\pi) = E_n + E_s$, and $concede(\pi) = L_e + L_w + L_{ew}$. The relation below then captures the notion that one payoff is worse than another if it leads to fewer tricks, or if it leads to the same number of tricks in a way that requires the declarer to establish more entries, or give away the lead more often.

Definition 8.2 For any two payoffs, π_1 and π_2 , $\pi_1 < \pi_2$ if:

$$\begin{aligned}
 &tricks(\pi_1) < tricks(\pi_2) && or \\
 &tricks(\pi_1) = tricks(\pi_2) \wedge entries(\pi_1) > entries(\pi_2) && or \\
 &tricks(\pi_1) = tricks(\pi_2) \wedge entries(\pi_1) = entries(\pi_2) \wedge concede(\pi_1) > concede(\pi_2)
 \end{aligned}$$

With this definition in hand, modifying the interpreter algorithm in the way described above enables us to generate a new form of payoff profile in which the set of possible

worlds is partitioned into sets (described by \mathcal{C} -conjunctions) that produce different payoff 8-tuples:

$$[\pi_1 \triangleright C_1, \pi_2 \triangleright C_2, \pi_3 \triangleright C_3, \dots] .$$

From these, it is a simple matter to generate profiles describing tricks by collecting together, for each possible number, n of tricks that can be won, all the C_i for which $tricks(\pi_i) = n$, and rewriting them as a new term. Similarly, for each of the resources that can be consumed/produced, the same process can be used to generate a profile of the possible worlds under which each amount of the resource will be required/generated.

These are the resource profiles which will form the basis for our higher-level reasoning about how individual lines of play will interleave. Now that we are able to generate them, we can go on to look at how they can be utilised in the interleaving process.

8.3 Refinement Search

In Chapter 3 we introduced refinement search as a unifying framework for comparing and contrasting partial order plan-space planning systems. We now return to this framework and use it to examine the possible architectures capable of carrying out the task of interleaving.

8.3.1 Review

Recall that refinement search is essentially a particular view of the generate-and-test paradigm. Rather than employing a search process that involves the gradual *construction* of a solution, refinement search starts with the *complete set* of possible sequences of operators, and then gradually subdivides it into smaller subsets by a process of repeated splitting (or branching, or *refinement*).

Kambhampati's generalised plan-space refinement algorithm is reproduced again in Figure 8.11 on the following page. By starting with a null set of constraints, the initial node describes (implicitly) the entire set of all the plans that can possibly be constructed. As constraints are added by successive calls to *Refine-plan*, the number

Algorithm *Refine-Plan*($\langle \mathcal{P}: \langle T, \mathcal{O}, \mathcal{B}, ST, \mathcal{L} \rangle, \mathcal{A} \rangle$)

Parameters: *sol*, the solution constructor function

G, the set of goals of the problem.

pick-prec, routine for picking preconditions from the plan agenda for establishment.

interacts?, routine used by pre-ordering to check if a pair of steps interact.

conflict-resolve, routine for resolving monotonic auxiliary constraint conflicts.

0 **Termination check.** If *sol*(\mathcal{P}, G) returns a solution, return it, and terminate. If it returns **fail**, fail. Otherwise continue.

1 **Refinement.** Refinements fall into two broad classes:

- **Establishment Refinement.** Refine the plan by selecting a goal, choosing a way of establishing that goal and optionally remembering the establishment decision:

1.1 **Goal Selection.** Using the *pick-prec* function, pick from \mathcal{P} a goal $\langle c, s \rangle$ to work on. *Not a backtrack point.*

1.2 **Goal Establishment.** Non-deterministically select a new or existing establisher step s' for $\langle c, s \rangle$. Introduce enough ordering and binding constraints, and secondary preconditions to the plan such that (i) s' precedes s (ii) s' will have an effect c , and (iii) c will persist until s (i.e., c is preserved by all the steps intervening between s' and s). *Backtrack point; all establishment possibilities need to be considered.*

1.3 **Book Keeping** (Optional). Add auxiliary constraints noting the establishment decisions, to ensure that these decisions are protected by any later refinements. This in turn reduces the redundancy in the search space. The protection strategies may be one of *goal protection*, *interval protection*, and *contributor protection*. The auxiliary constraints may be one of point truth constraints or interval preservation constraints.

- **Tractability Refinements** (Optional). Help in making the plan-handling and consistency check tractable. Use either one or both:

1.4 **Pre-ordering:** Impose additional orderings between every pair of steps of the partial plan that possibly interact. according to the static interaction metric *interacts?*. *Backtrack point; all interaction orderings need to be considered.*

1.5 **Conflict Resolution:** Add orderings, bindings and/or secondary (preservation) preconditions to resolve conflicts between the steps of the plan, and the plan's auxiliary candidate constraints. *Backtrack point; all possible conflict resolution constraints need to be considered.*

2 **Consistency Check** (Optional). If the partial plan is inconsistent (i.e., it has no safe ground linearisations), fail. Else continue.

3 **Recursive Invocation.** Call *Refine-Plan* on the refined plan.

Figure 8.11: Generalised plan-space planning algorithm (repeated)

of *solution candidates* at each node is reduced until `sol` can return an answer. The exact behaviour of the algorithm is determined by the particular way in which the steps of goal selection and establishment, book-keeping, and tractability refinements are specified. We have already seen how Kambhampati describes existing plan-space planners as the results of particular instantiations of these slots: we now look at how we can cast interleaving in the same framework.

8.3.2 Interleaving as Refinement Search

The overall task of interleaving is to take a set of lines of play for individual suits and produce a line of play for an entire game. Here, we interpret this as a problem in finding the appropriate constraints on the allowable orderings between the operators in each line of play.

Initially, with no constraints, any possible ordering of the constituent operators is allowed. The aim of interleaving is to reduce this number of possible combinations to the point where the success or otherwise of the interleaving can be determined.

Here, we will informally characterise FINESSE's interleaving algorithm in terms of refinement search. In the following sections we then give substance to this basic outline. The basic instantiation made by interleaving of the refinement framework is as follows:

- **Solution candidates.** Lines of play for an entire game.
- **Agenda.** The only operators to be considered are those contained in the lines of play to be interleaved, and the overall aim is to determine whether this interleaving is possible. The agenda therefore contains any resource requirements of the operators that have yet to be satisfied.
- **Goal Selection.** Selection of an unsatisfied resource from the agenda, based on a likelihood heuristic (detailed in §8.4)
- **Goal Establishment.** We assume that no extra operators can be added (*i.e.*, we assume that we are interleaving *given* lines of play), establishment consists of introducing ordering constraints on the suppliers to ensure they occur in the correct position. (See §8.5.)

- **Book-keeping.** Contributor protection via the posting of auxiliary constraints. (See §8.6.)
- **Tractability refinements.** Assume a total ordering. The number of possible orderings of the operators within the input plans may not be greatly reduced by the imposition of resource-based constraints, and the number of combinations is typically too large to examine explicitly. A total ordering refinement is therefore used to enable the solution constructor function to treat all allowable orderings as equal and to tackle the problem of non-locality. (See §8.6.)
- **Consistency check.** Plans with un-suppliable resources are pruned.

As we suggested in the introductory section, the main source of refinement will be resource-based reasoning. Exactly how this reasoning will function is described in the following sections. Before beginning this development, however, let us first present in Figure 8.12 a simplified form of the generic planning loop of Figure 8.11.

If `sol` cannot determine whether an interleaving will succeed, and there are goals (resource requirements) left unsatisfied:

- Select a goal to be established. Not a backtrack point.
- Non-deterministically select a new or existing establisher step to achieve selected goal. Backtrack point.
- Impose appropriate constraints, carry out appropriate book-keeping, tractability refinements, consistency checks.

Figure 8.12: A simplified refinement planning algorithm applied to interleaving

This figure provides a simple basis upon which we will elaborate as we explore the details of the algorithms which instantiate each slot. Over the next few sections we will examine the details of goal selection, goal establishment, the posting of auxiliary constraints, and the solution constructor function, until we have an overall view of how the interleaving process as a whole functions.

8.4 Goal Selection and Establisher Selection

In the above outline of an interleaving architecture, the first tasks that are encountered are goal selection and establisher selection. Here we detail how the resource profiles described earlier in this chapter can be used to implement this.

8.4.1 Finding a Solution vs Finding the Best Solution

According to the simple planning loop of Figure 8.12, once all the goals in an interleaving task have been satisfied, the solution will consist of a set of constraints determining a (not necessarily unique) ordering on the operators of the interleaved plans. However, in a domain with uncertainty, this approach faces the difficulty that in general it may not be possible to produce an interleaving where *all* the goals are completely satisfied. Rather, each goal may be satisfied only under some subset of the possible worlds, so that the global plan has a probability of success between 1 and 0. This means that unless we find a plan which always succeeds, there is no longer any easily identifiable termination criterion: we may have to examine *all* the possible establishment possibilities before we can be sure that we have found the best. Clearly, this is potentially very inefficient, and in Chapter 10 we shall see that when the desire is to select between different sets of plans, rather than just finding the best possible interleaving of any one of them, it may be wasteful to consider all the possible interleavings of any one set. For example, it may be possible to establish that *all* the interleavings of one set are inferior to those of another set. FINESSE therefore uses a heuristic to determine the choice of the goal selection and establisher steps, in an attempt to generate the ‘good’ interleavings first. This heuristic is based on the probabilities associated with the resource consumption and production of the plans being interleaved.

8.4.2 Using Probabilities

In §8.2.2, we saw how FINESSE’s interpreter and subsumption algorithm could be used to produce resource profiles of a line of play. In particular, for any particular resource we saw that it was possible to form a resource profile detailing either its consumption or production as a list of pairs of the form

$$[N_1 \triangleright C_1, N_2 \triangleright C_2, N_3 \triangleright C_3, \dots] , \quad (8.1)$$

where the N_i are distinct integers greater than zero in ascending order and the C_i are terms describing disjoint possible worlds. For a given line of play, each type of resource consumed can be described by a profile in which the N_i represent possible amounts of consumption and the corresponding C_i describe the worlds under which each amount of the resource is required. Similarly, the resources supplied are described by profiles which describe the worlds under which each possible amount of supply is generated.

Given a collection of profiles describing the resource analysis of some lines of play that are to be interleaved, it is possible to rank the resource suppliers and consumers according to how *likely* they are to produce or consume their resources. For example, the probability that the resource profile of (8.1) describes the consumption (or production) of *at least one* unit of resource is $\text{prob}(C_1 \vee C_2 \vee \dots)$, which, since all the C_i are disjoint and all the N_i are greater than zero, can be written as:

$$\sum_j \text{prob}(C_j) .$$

One way to modify the planning loop of Figure 8.12 on Page 206, then, is to replace the random selection of goals and establishers by an algorithm which selects the most *likely* (in terms of probability) consumer and supplier at each stage. The output of the first loop of this planning algorithm will clearly not in general be the best amongst all the possibilities, but in Chapter 10 we will argue that it is a useful way of identifying which plan set from amongst a number of alternatives warrants further consideration.

Before discussing how the resource profiles can be maintained when using such a heuristic, we should point out that we will be exploiting the feature that the operators to be interleaved are all known in advance, by virtue of being present in the lines of play being interleaved. In other situations, a different algorithm may well be more appropriate. For example, in problems where further operators (with attendant resource requirements) may be introduced at a later stage, a least commitment strategy on the resource suppliers may be preferable, so that operators introduced to the plan at later stages would have the maximum possible chance of being suppliable.

8.4.3 Maintaining the Profiles

Let us assume that we have chosen (based on the probabilities) the most likely consumer and producer of a given resource, and that their respective profiles for this resource are:

$$\begin{aligned} \text{consumer: } & [N_1 \triangleright P_1, N_2 \triangleright P_2, N_3 \triangleright P_3, \dots] \\ \text{producer: } & [M_1 \triangleright Q_1, M_2 \triangleright Q_2, M_3 \triangleright Q_3, \dots] . \end{aligned} \quad (8.2)$$

If the probabilities associated with each of these profiles is 1 (*i.e.*, there is always at least one resource), then the act of assigning a single unit of resource from the supplier to the consumer would reduce both supply and demand in any possible world by one. Thus, the new profiles would be

$$\begin{aligned} \text{consumer: } & [(N_1 - 1) \triangleright P_1, (N_2 - 1) \triangleright P_2, (N_3 - 1) \triangleright P_3, \dots] \\ \text{producer: } & [(M_1 - 1) \triangleright Q_1, (M_2 - 1) \triangleright Q_2, (M_3 - 1) \triangleright Q_3, \dots] . \end{aligned} \quad (8.3)$$

However, it may be that the resource production is conditional, *i.e.*, that $\text{prob}(Q_1 \vee Q_2 \vee Q_3 \dots) < 1$. To examine how this circumstance would affect the consumer, let us examine a single entry, $N_i \triangleright P_i$, say, of its profile. If the resource supply is successful — *i.e.*, if both $(Q_1 \vee Q_2 \vee Q_3 \dots)$ and P_i are true — then only $(N_i - 1)$ further resources will be required. Thus, the original term from the consumer profile should be replaced with:

$$(N_i - 1) \triangleright P_i \wedge (Q_1 \vee Q_2 \vee Q_3 \dots).$$

However, there is now also the chance that the resource supply will be unsuccessful. In this case, N_i resources will still be required. Thus, the following term should also appear in the updated resource profile:

$$N_i \triangleright P_i \wedge \neg(Q_1 \vee Q_2 \vee Q_3 \dots).$$

The general form of the updated consumer profile when supplying one unit of resource under conditions $Q = Q_1 \vee Q_2 \vee Q_3 \dots$ is therefore:

$$\begin{aligned} \text{consumer: } & [(N_1 - 1) \triangleright P_1 \wedge Q, N_1 \triangleright P_1 \wedge \neg Q, \\ & (N_2 - 1) \triangleright P_2 \wedge Q, N_2 \triangleright P_2 \wedge \neg Q, \\ & (N_3 - 1) \triangleright P_3 \wedge Q, N_3 \triangleright P_3 \wedge \neg Q, \\ & \dots\dots\dots] . \end{aligned} \tag{8.4}$$

A similar situation arises when considering the effect on the resource supplier's profile when the resource consumption is conditional. Again there are two possible situations: one where the resource is utilised and one where it is not. If we consider a single entry of the supplier profile, $M_i \triangleright Q_i$, the cases are similar to those for the resource consumer.

$$\begin{aligned} & (M_i - 1) \triangleright Q_i \wedge (P_1 \vee P_2 \vee P_3 \dots) \\ & M_i \triangleright Q_i \wedge \neg(P_1 \vee P_2 \vee P_3 \dots) . \end{aligned} \tag{8.5}$$

This suggests the new planning loop detailed in Figure 8.13, where both resources and consumers are chosen according to likelihood, and the resource lists maintained as above.

8.4.4 The Effects of Ordering

We have not yet discussed the exact form of the constraints that will ensure that resource suppliers occur in the desired positions (this will be the subject of §8.5). However, we do need to be aware that the presence of such constraints between parts of a plan set may affect the assignment of resource suppliers.

For example, the simple situation of Figure 8.14 depicts a set of plans with just three components: a, b, and c. One of these, b, is constrained to come after a, whilst c has no ordering constraints. We will assume that there is just one resource in the domain, of which both a and b consume one unit and c produces one unit. However, the resource

1. Make a list of profiles for each resource required, and order by probability. Do the same for resource production.
2. While there is still outstanding (*i.e.*, probability greater than 0) resource consumption:
 - Select the most likely resource consumer. Not a backtrack point.
 - Non-deterministically select an establisher step, starting with the most likely. Backtrack point.
 - Maintain the consumer profile by applying (8.4), and the supplier profile by applying (8.5).
 - Impose appropriate constraints, carry out appropriate book-keeping, tractability refinements, consistency checks.
3. Run `sol` to find probability of successful interleaving.

Figure 8.13: A revised algorithm for resource-based interleaving

supply and consumption is conditional: **a** is a consumer under some condition P , **b** is a consumer under condition Q , and **c** is a producer under condition R .

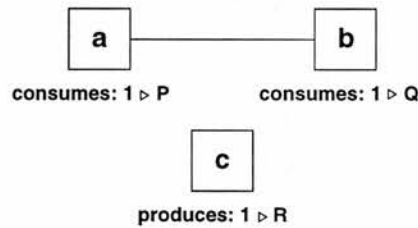


Figure 8.14: A plan set with resource requirements and an imposed ordering

It is fairly easily demonstrated that the different legal orderings of **a**, **b** and **c** will have different chances of success. For instance, if we order **c** to come last, the plan will only be executable if it transpires that neither **a** nor **b** require resource supply (*i.e.*, if $\neg P \wedge \neg Q$ is true). Alternatively, the plan's chances of success can be increased by constraining the resource supply of **c** to be available for one of **a** or **b**. For instance, if we attempt to supply **b**, then **a** will only be executable if its resource is not required (*i.e.*, if $\neg P$), and **b** will be executable both when it requires no resource ($\neg Q$) or when it consumes a resource from **c** ($Q \wedge R$). Thus, the plan set will be executable under

the conditions

$$\neg P \wedge (\neg Q \vee (Q \wedge R)). \quad (8.6)$$

If, on the other hand, we supply *a*, then *a* will be executable under the conditions $\neg P \vee (P \wedge R)$, and *b* will be executable whenever it requires no resource (*i.e.*, if $\neg Q$) or whenever *a* did not consume *c*'s resource supply. Thus, this ordering is successful under the conditions

$$(\neg P \vee (P \wedge R)) \wedge (\neg Q \vee (Q \wedge R \wedge \neg P)). \quad (8.7)$$

It is a simple matter to show that the formula of (8.7) implies that of (8.6) and that the reverse is not the case. Thus, the option of supplying *a* is superior to that of supplying *b*, since it gives a better probability of success. The generalisation of this is that, given a choice of (conditional) goals to establish, it is best to begin with those that have already been constrained to occur towards the front of the plan, since any unused supply will then be available for succeeding operators. Also, given a choice of suppliers with which to supply a goal, it is best to choose those which have already been constrained to occur towards the start of the plan, since this will leave other suppliers free for the later operators.

Before incorporating this into our planning loop, however, let us stop to consider the possible implications of ordering constraints on our resource profiles. When dedicating one unit of supply from a supplier to a specific consumer, the supplier's resource profile is modified as in (8.5). If it transpires that the consumer does not make use of this supply, then we may indeed be able to make use of it somewhere else in the plan. However, as we have seen above, this will *only* be possible if the new position we choose for the supplier is *after* the original position. Thus, the type of modification to the profile of the resource supplier suggested by (8.5) will over-estimate the resource suppliable to operators which are in front of the resource consumer. Since this may lead to unsoundness (the planning loop may believe that resources have been supplied to all consumers when in fact this supply may be dependent on an implicit ordering) it is best to modify the supplier profiles with the more simple (8.3) which ignores the possibility that an assigned resource can be utilised at a later point in the global

plan if its consumer does not utilise the supply. This approach will ensure that the available resource supply will always be either exactly measured or under-estimated, and is therefore a more sound, conservative basis for our heuristic.

Taking into account the above arguments, then, we can re-write our planning loop as in Figure 8.15.

-
1. Make a list of profiles for each resource required, and order by probability. Do the same for resource production.
 2. While there is still outstanding resource consumption:
 - Select the consumer occurring nearest the front of the plan. If this is not unique, resolve ties based on likelihood of consumption. Not a backtrack point.
 - Non-deterministically select a supplier, starting with the those nearest the front of the plan, and again resolving ties on the basis of likelihood. Back-track point.
 - Maintain the consumer profile by applying (8.4), and the supplier profile by applying (8.3).
 - Impose appropriate constraints, carry out appropriate book-keeping, tractability refinements, consistency checks.
 3. Run `sol` to find probability of successful interleaving.

Figure 8.15: A final algorithm for resource-based interleaving

8.5 Auxiliary Constraints and Book-keeping

The steps of ‘Imposing appropriate constraints...’ and ‘Running `sol`...’ in the updated planning loop of Figure 8.15 have not yet been described. The remainder of this chapter will therefore address these issues, starting here with a discussion of how to post the appropriate constraints. This will take us back to look at the partial plan representations first introduced in Chapter 3 during our review of the planning literature.

8.5.1 Partial Order Plan-space Representation

We have seen that in an action-ordering representation the operators of a plan are represented as nodes, and the ordering relations between them as arcs joining the nodes. Further, we have seen that partial order plans do not require the orderings between the operators to be fully specified and that this can allow a large set of plans to be represented without explicitly presenting each one separately.

Figure 8.16 shows how the simplified lines of play which we introduced in Figure 8.1 on Page 188, could be represented using the type of partial plan representation introduced in Chapter 3. As it stands, this network constrains all the operators in the individual lines of play to occur in sequence, but allows all possible orderings of the actions between the lines of play. The aim of posting auxiliary constraints is to guarantee that the possible orderings will be executable.

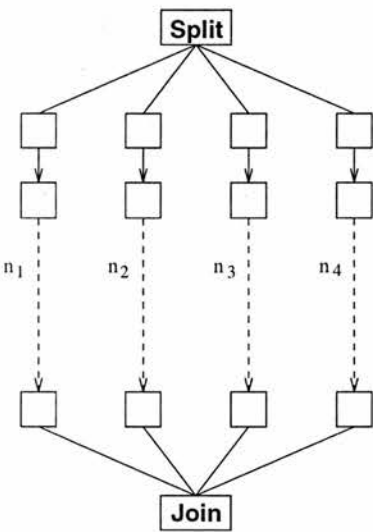


Figure 8.16: Combining lines of play in a network

Recall that the type of linear line of play illustrated here is an over-simplification of the general structure, which usually contains disjunction over which we have no control. This complication has led us in this chapter to describe the possible dependencies between individual lines of play in terms of the resources they consume or produce. We therefore treat individual lines of play *abstractly* as individual entities described purely

by their resource profiles. The nodes in our partial plan representation will therefore not represent individual operators, but lines of play composed of these operators.

For example, consider the deal of Figure 8.17 where the contract is 3 No Trumps and West’s lead is the 10♥.

Contract: South – 3 No Trumps
Lead: West – 10♥

♠ Q J 5
♥ K 8
♦ 7 6 2
♣ 8 7 6 5 4

N
W E
S

♠ A 6
♥ A 4 2
♦ A K Q J 10
♣ J 10 9

Figure 8.17: An example Bridge deal

The first action of a human declarer faced with this situation would typically be to count the top tricks. In FINESSE, this effect is achieved by restricting the interpreter algorithm to consider only the cash and sequence tactics when looking at each suit, resulting in the following plans, which (assuming they can be interleaved) produce a total of eight certain tricks:

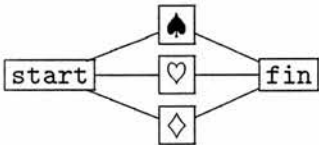
Suit	Brief Summary of Plan
♠	cash South’s Ace.
♥	cash the Ace and King.
♦	cash South’s five top tricks.

Rather than look at how these simple lines of play might be interleaved, let us examine the more interesting case where one of them is improved in an attempt to supply the extra trick needed to make the contract. Since the spade suit can be made to supply a certain extra winner by cashing the Ace and then forcing out the King by playing the Queen, we will replace the original line of play in this suit by this improved version.

The resource analyses of these lines of play can then be found as described in §8.2.2, producing the information contained in the following table:

Suit	Resources Required	Resources Produced
♠	entry(n,1)	lead(n,1), lead(s,1), lead(e,1) $\triangleright P$, lead(w,1) $\triangleright Q$
♥	-	lead(n,1), lead(s,1)
♦	-	lead(s,3)

where P and Q describe the condition that East or West, respectively, holds the $K♠$. Rather than representing each line of play in its entirety, we choose to simply create a node in our partial plan representation for each one. This may be represented using a network as follows:



As it stands, this network allows all orderings of the tactics in the lines of play, with the proviso that the temporal constraints implied by the tree structure of each individual line of play itself is adhered to. It is, of course, not true that all these orderings are valid, since we have already seen that the spade suit has an unsatisfied resource requirement. The planning loop of Figure 8.15 will select this consumer (as it is the only one) and attempt to supply it. The only outside suit which is capable of supplying the resource is the heart suit, so we will wish to modify the partial plan to ensure that (the resource-supplying part of) the line of play in hearts will always be available to satisfy the resource requirement of the spade suit. Below, we consider how this can be done.

8.5.2 To Split or Not to Split?

When constraining resource supply, it turns out that it is important to be able to identify not only which nodes produce resources, but also which *parts* of the plans that the node represent are responsible for the production. For instance, in the example above, it is possible to imagine splitting the ♥ node into two portions: one supplying the entry required by the ♠ node, and the second representing the ‘remaining’ operators.

In this particular case, we can identify the cashing of the $K\heartsuit$ as the operator in the line of play for the heart suit which generates the required resource. However, in general it is not always so easy to split away a resource supplying portion of a line of play from its remainder. This is because typically, the resource supplying portion may only be executable at certain points within the line of play, and the tree-like structure makes it difficult to be specific about the way the line of play will behave when split. For instance, consider the simple line of play in Figure 8.19 on the next page (recall that a line of play is generated in a state-based manner, so it is the *arcs* in this figure that correspond to operators).

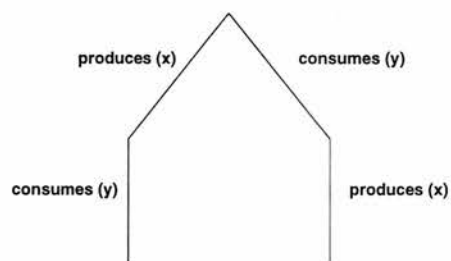


Figure 8.18: Branching in a line of play

Imagine that this figure represents a line of play in which there are two possible courses of action (we will ignore for the moment the issue of whether it is MAX or MIN that controls this branching). If we wish to use the operators in this line of play to supply a resource x , what is the resource analysis of the ‘remainder’ of the plan which is not involved in the resource supply? The correct answer to this question involves looking at each possible branch of the plan separately. If the left-hand path is followed, there will be one operator left *after* the supplier, and that operator will consume resource y . However, along the other branch this operator will have to come *before* the supplier.

Another effect of this possible branching is that the resource analyses of the new nodes when a supplier is split may depend on the way in which the split is carried out. For example, consider again the task of supplying the resource x , but this time from the new plan of Figure 8.19 on the following page. Here, if the left-hand path is followed, the supplier of x will be followed by an operator that supplies resource y . However, along the other branch this operator will also have a resource consumption (of the

resource **z**), which on the first branch was consumed by the supplier of **x**.

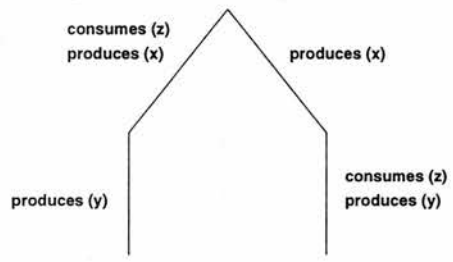
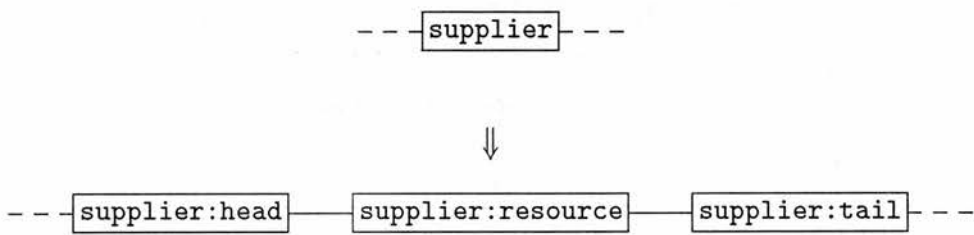


Figure 8.19: Another example of branching in a line of play

In general, then, when splitting a node being used as a **supplier**, each possible branch within the supplier will decompose into an operator responsible for the supply of the **resource** concerned, possibly accompanied by the **head** of the plan which precedes the **supplier** along that branch, and the **tail** which follows it.



Since a line of play may contain a large number of branches, there will be many ways in which a **supplier** node may be split. Rather than one network describing the result of splitting a node then, we are faced with the possibility of having to work with multiple networks; one for each of the (perhaps large) number of ways in which each supplier node might be split.

To avoid this likelihood of dealing with many possible sets of constraints in the planning loop, we retain the notion of representing each line of play by a *single* node, without conducting any splitting. To add a constraint into the network, we simply insert a term of the form:

```
interleave(Consumer, Supplier, Resource)
```

in the links of both the **Consumer** and the **Supplier** concerned. The actual examination of the possible ways in which the supplier may split is then carried out by the solution

constructor function, which uses the `interleave` constraints to ensure that resource consumers occur in the correct positions.

Checking Orderings

FINESSE carries out a consistency check when adding constraints to a partial plan, to ensure that it is not imposing an impossible ordering on a supplier's actions. For instance, in Figure 8.19 it is possible to generate the resource `x` followed by the resource `y`, but not `y` then `x`. If, for example, we are already using the line of play to supply the resource `x`, then, we cannot use it again to supply `y` to an operator that occurs before the consumer of `x`. A check is therefore carried out when posting any constraint that the ordering in which the supplier is required to generate resources is at least present on one of the branches within the structure of its line of play.

8.5.3 The Effect of an Opposition

In the domain of Bridge, there are almost always points during the play of a hand at which the opposition will control the next action. This gives rise to a major problem which we have not yet tackled: despite constructing a network constraining the ordering of suppliers and consumers, we may not be able to ensure that these constraints will be followed; the opposition may be able to break the constraints when it is their turn to dictate the play. For example, consider the situation of Figure 8.20 on the next page, where it appears to be possible for the resource requirements of the left-most line of play to be satisfied by the resource productions of the other two.

As it stands, constraining the production of the resources `x` and `y` to interleave with the execution of the first line of play will indeed guarantee a successful interleaving. However, consider the situation where the first operator of the left-most line of play (which currently has no resource consumption or production) has the effect of giving the control over the next action to the opposition. If the defenders play to the best of their ability, they will be able to prevent the interleaving from succeeding by immediately forcing the next action to come from the right-most line of play. This will result in the resource `y` being produced too soon to be utilised by the operator for which it was

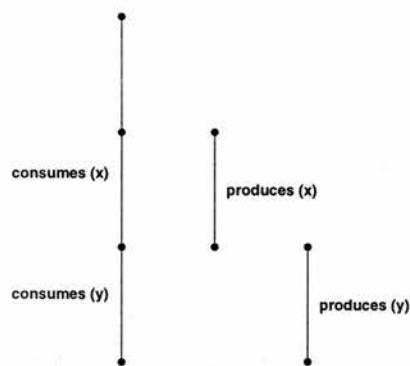


Figure 8.20: Lines of play fragments that can be interleaved

intended, thus making it impossible to execute the global plan.

In general, there may be more than one portion of an interleaving which the opponents can disrupt in this way. This means that an attempt to simply ‘protect’ a portion of an interleaving by using further suppliers to ‘over-supply’ its consumption will not be guaranteed to succeed, because the defenders will be able instead to disrupt another portion of the interleaving which would have been executable if protected by the same over-supply. Each time the opposition may possibly be given the lead, then, we should be considering the best way to assign any over-supply to *each particular* disruption that the defenders may have made. So, this problem introduces further possible disjunction and calls for more separate sets of constraints to deal with each of the possible ways in which the world might turn out.

Rather than explicitly creating separate sets of constraints for each disjunctive possibility, we resort to pushing the burden of examining all the possibilities into the solution constructor function. We examine this in the following section.

8.6 A Solution Constructor Function for Bridge

In this section we present an algorithm which can use the constraints described above to determine which tactics to select when executing a global plan. This algorithm will therefore be able to answer the question of how well a set of lines of play will fit together, given a particular assignment of producers to consumers.

As we have mentioned already, FINESSE uses a total ordering tractability refinement when interleaving given lines of play. This assumes that all the possible tactic orderings that meet the constraints are equal, or equivalently that the constraints specify *all* the possible interactions between the lines of play being interleaved. This refinement both releases us from the need to examine all the possible tactic orderings that meet the constraints, and also means that the effects of non-locality will not arise, since we never choose between tactics with possibly different results.

Below, we describe FINESSE's actual solution constructor function. First we assume the presence of a method for selecting a single tactic that meets the interleaving constraints at any point, and describe how actual lines of play composed of these tactics can be constructed. The actual algorithm used by FINESSE for this task is rather complicated, so to help in understanding it, we describe its features incrementally, building gradually from the simplest conceivable architecture by making successive, efficiency-motivated modifications. After this development, we show how a network of constraints can in fact be used to determine a unique tactic to select at each point.

8.6.1 A Naïve Approach

Given a method for selecting tactics which obey the interleaving constraints, the most immediately obvious way to determine the chances of success of an interleaving is first to fix attention on one particular world. From the initial state in this world it is possible to construct a line of play explicitly by selecting a tactic, generating all the possible MIN responses, and then repeating the process for the resulting states, either until no further tactics are applicable or until thirteen tricks have been played. The number of tricks that can be expected against best defence in that world is then simply the minimum of the number of tricks gained at the leaf nodes of the tree. By repeating this procedure in each of the possible worlds a profile can be produced. Figure 8.21 depicts a version of this type of search for a domain with possible worlds w_1, w_2, \dots, w_n .

It should be clear that in any domain with a large number of possible worlds, this approach is not going to be very efficient. In our particular case there are ${}^{26}C_{13} \approx 10^7$ possible worlds to examine. In each of these, the tree of possibilities is 13 tricks deep and rather bushy, since the basic branching factor is 26 for the first defenders'

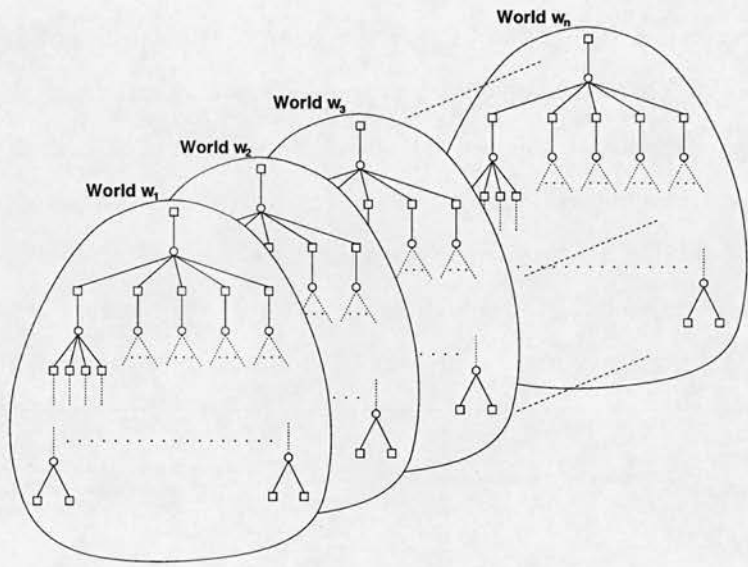


Figure 8.21: An interleaving strategy involving constructing a line of play for each possible world

play, 25 for the second, and so on. Clearly, the requirements of legal play will reduce this branching factor somewhat, and a further efficiency improvement can be made by utilising the single-suit planner’s specification of cards into critical and low card sequences to further restrict the branching factor. However, this does not address the basic problem that the large number of possible worlds will make any interleaving procedure following this approach intolerably slow.

8.6.2 Utilising an Uncertainty Representation Language

In the previous chapter we described an uncertainty language capable of representing the uncertainty in the Bridge domain compactly, and an efficient technique for manipulating terms in this language. As in the case of single-suit planning, where using this language to represent information qualitatively resulted in efficiency improvements, we can reduce the search required by the naïve algorithm described above by representing multiple possible worlds with single expressions. A simple way to incorporate this technique into the algorithm already described is to first of all form a single term to represent all the possible ways in which the cards may be distributed between the defenders at the beginning of the play. A single search tree can then be construc-

ted by generating branches that represent possible plays which take into account this uncertainty (*i.e.*, rather than fixing the state of the world and generating plays from the defenders' known hands, each defender may play any legal outstanding critical or low card). Clearly, it will not be possible to follow each branch of the search space under each possible world, so as the search progresses, the initial set of possibilities is gradually subdivided into smaller sets. Figure 8.22 illustrates this process pictorially for an interior node of the search space. Note that in general, each possible world at the original node will also be present in at least one of the daughter nodes.

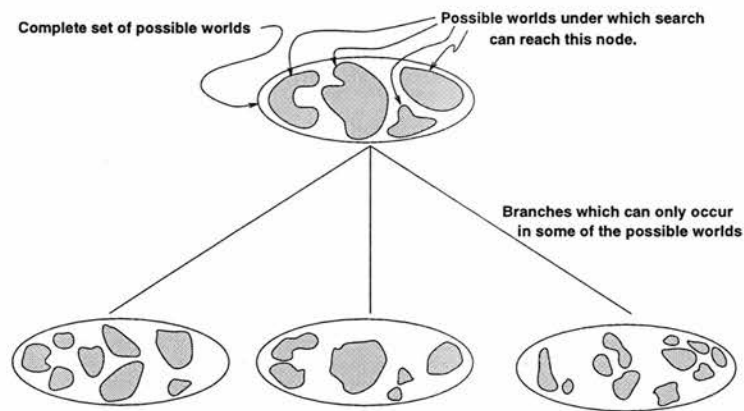


Figure 8.22: Pictorial representation of the use of uncertainty terms to improve search efficiency

The number of tricks won by the declarer at each leaf node of the space becomes a possible result of the interleaving under *each* of the possible worlds under which the leaf node can be reached. The minimisation step is now replaced by a subsumption check where for each world we assume that MIN will direct play to one of the leaf nodes where the declarer's return is smallest in that world.

8.6.3 Redistributing the Search Burden

It is sometimes possible to improve the performance of search algorithms by moving some of the search from inefficient to efficient areas. In our case, we have already seen that manipulation of uncertainty terms is more efficient than repeatedly generating similar search trees. A further improvement using this same principle can be made,

this time by removing some branches of the search space into the uncertainty terms.

The single-suit lines of play are composed of single-suit tactics. When considering the possible replies by the defence to such a tactic in the interleaving stage, we can utilise the same approach as the single-suit planners to reducing the branching factor. That is, all the cases where one defender shows out in the suit can be represented by the single case of that defender being void. Clearly, this is a slight over-simplification, as eventually it will be necessary to separate out the cases in which the defenders win tricks by ruffing. However, as a developmental modification it will prove to be a useful tool.

The result of this modification is that the actual search space is kept more compact, but at the cost of larger uncertainty expressions. However, the manipulation of these expressions is relatively efficient.

8.6.4 A State-based Search Alternative

One final improvement can be made by realising that there is an alternative to maintaining uncertainty terms as the search tree is generated. Rather than start the search with a term describing the complete set of worlds, we can instead start with a representation of the initial state detailing the cards which are remaining. With this representation, descending a branch will simply involve the removal of the appropriate cards from the current state, rather than the updating of an uncertainty expression. When the leaf nodes of the search are reached, it is assumed that *any* of the possible distributions of the remaining cards could be true (unless one defender is already known to be void) and an uncertainty term constructed accordingly. This term is then passed back up through the tree, updating it as necessary to take account of the cards played by each defender.

Clearly, this approach is very similar to the original one of gradually subdividing the original worlds, and indeed in the single-suit case there is very little performance difference between them. FINESSE actually uses the state-based approach in the single-suit case since the expressions it forms at the leaf nodes of the tree have to describe only a subset of the outstanding cards and are therefore small. They will increase in size

as they are passed back up through the tree and incorporate gradually more plays of cards, but whilst they are smaller in the lower regions of the search space, processing time is saved.

However, in the case of interleaving there is a more significant benefit to be gained by using a state-based approach. This is mostly due to the redistribution of the search burden described in the previous section. By allowing branches which are essentially variables in the search space we saw that the branching factor could be reduced at the cost of extra work being carried out to maintain the uncertainty expressions. However, these branches are not true variables, since every possible instantiation of them is considered when generating the daughter uncertainty terms of each node. With a state-based approach, however, it is possible to treat these branches as true variables whose values are left unknown. Rather than trying to consider which cards a defender may have chosen when he has no cards in the suit led, a flag is simply set indicating that he is void. Along any path of the search space in which a defender is flagged as being void in some suit, the uncertainty terms generated at the leaf node only describe worlds where the remaining cards in that suit are held by the other defender. In this way the actual cards played by the defender who was void are never explicitly considered, significantly increasing the efficiency of the algorithm.

8.6.5 Using a Network of Constraints to Guide Interleaving

Given the above approach of a state-based search over the possible worlds, what remains is the ability to use a partial plan containing a set of constraints to select a tactic at each point. There are clearly a number of obvious properties which must be checked, such as whether a node's preconditions are met and whether a node is actually at the front of the net. However, there are also some more subtle features which are introduced individually below.

Node Preference

Since a partial plan may in general contain actions which are unordered with respect to each other, we may often be faced with arbitrary choices. However, we have already

seen that to avoid having to consider *all* the possible orderings and to side-step the problem of non-locality, we wish to interpret a network as imposing a *single ordering* on the plan execution (*i.e.*, that all constraints have been explicitly represented).

Thus, we have a different situation to that of single-suit planning, where we allowed the planner to consider all the possible different orderings of applicable tactics and then select the best. The single-suit planner was able to use this ability to discover for itself the situations where, for example, it was best to order a finesse before a cash, or a cash before a finesse. Our tractability refinement for interleaving, however, means that the interleaving architecture will not have the opportunity to ‘discover’ whether some orderings are superior by testing them. It is therefore necessary to have a technique for explicitly representing some hitherto implicit Bridge constraints, such as the desirability of drawing trumps as soon as possible in a suit contract.

In the case of drawing trumps, it would not be desirable to simply constrain the trump suit node to come before all the other nodes, as this may make it impossible to satisfy the resource requirements of the network. Instead we require some notion of *preference* which is less important than actual network ordering, but allows us to prefer some nodes over others within the constraints imposed by the resource reasoning.

In FINESSE, this is achieved by assigning each node a numerical preference greater than zero. Decisions at points where multiple nodes may be selected are then resolved by simply choosing the nodes in order of preference.

Choosing the order in which to execute the branches of split nodes could be informed by a number of factors. For example, wherever possible in a suit contract, the trump suit should be as uncommitted to interleaving duties as possible, to enable trumps to be drawn quickly. Also, there are a number of general playing practices which humans identify as ‘making good sense’:

- If the declarer has no means of preventing the defence from taking enough top tricks to break the contract, control should not be surrendered until the contract is made.
- Playing in a suit in which declarer or dummy is void (thus forcing a discard) may weaken declarer’s control in the discarded suit. It therefore may be best

to position such tactics after any other plays that may surrender control to the defence.

- Similarly, the cashing of declarer's winners in a suit may have the side-effect of establishing masters for the defence. If this is the case, these tactics should also be delayed until plays that surrender control to the defence have been made.

For instance, in our example of Figure 8.17, playing on the diamond suit will not establish any winners for the defence, but it will force the declarer to find discards from the North hand (probably in the club suit). Since the line of play for the spade suit involves giving the lead to the defence, we should therefore consider executing the tactics in this branch before those in the \diamond node. Also, the $K\heartsuit$ is the stop card that prevents the defence from taking enough heart tricks to break the contract, so the lead should not be surrendered once it has been played.

Ensuring Resource Supply

The semantics of the **interleave** constraints are that not only should the resource be constrained to occur during the execution of the consumer, but it should also occur *at the correct time*. Since in general the resource consumer may have actions at its head which may not preserve a supplied resource, the supplier should not be used to generate its supply until the consumer actually requires it. When selecting a resource supplier for execution, therefore, we must either check that it has spare, uncommitted supply, or that at least one of the consumers to which its supply is dedicated requires the resource immediately. (Note that it is not possible to control this process by 'only picking resource consumer nodes' for execution, because consumers can also be suppliers for other nodes.)

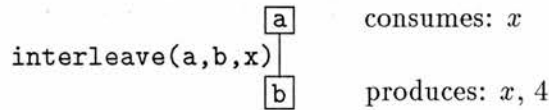
Bookkeeping

After an operator is selected, various bookkeeping measures can be taken. For example, if the tactic was the final step in a compound action, the empty node remaining can be deleted from the network of constraints. Also, any links which constrain other actions

to come after the (now empty) node can be removed.

In addition to such optional steps, however, there are some situations, such as the execution of a resource supplier, in which bookkeeping steps *must* be taken. If the *intention* of executing a supplier tactic is to supply a resource for a particular consumer, and this tactic actually results in the successful supply of this resource, the supply must be protected so that the intended consumer can utilise it. The protection is therefore signalled, constraining all subsequent actions to conserve the resource until the consumer is executed. This is done by posting a constraint of the form `protect(Consumer, Supplier, Resource)`. This constraint is only removed when the `Consumer` utilises the supplied resource. Whilst it is in force, no other operator may delete or supply the `Resource`. Thus, this is a form of contributor protection.

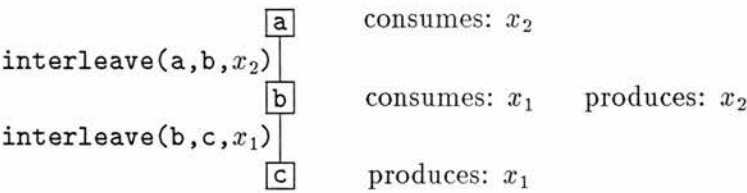
Notice that here it is actually important to distinguish whether the supply of a resource is intentional or not. To see this, consider the following situation



where there are two nodes, `a`, and `b`, and node `b` supplies four units of resource. If node `a` has the higher preference it will always be selected over node `b`, until it requires its resource consumption, which node `b` will then be called upon to supply. If, however, it is node `b` that has the higher preference, we will want to execute tactics from the front of it until there is only one unit of resource remaining, which must then be kept until the execution of tactics from node `a` reaches the point at which the resource is required. Thus, there are situations, when the producer contains *uncommitted* supply, in which the generation of a resource will not mean the subsequent execution of its consumer. It is therefore important to record the context in which actions are selected. *FINESSE* does this by imposing an interleaving constraint of the form `protect(Consumer, Supplier, Resource)` only when `Supplier` has successfully supplied a resource for the explicit consumption of `Consumer`.

This scenario can be extended to include more than two nodes. For example, in the example below there are now two resources x_1 and x_2 . In order to execute node `a`, we

will have to use the resource supplier in node *b*, which in turn requires the execution of node *c*. When selecting *c*, then, we need to remember not only that its resource will be used by *b* but also that when *b*'s resource is produced it will be consumed by *a*. In *FINESSE*, this is done by maintaining a *context* which records the resource protections and also the nodes (if any) which may consume any *subsequent* resource supply.



Schematic Representation

The flow chart of Figure 8.23 on the following page depicts pictorially how the above considerations affect the use of a network to interleave compound actions. Here, the three rectangular boxes represent decision procedures capable of generating backtrack points until a choice which is successful is found. If no executable tactic obeying the constraints of the network is found, it may be possible to continue with the execution by supplying a resource that was unused (and therefore now left free) by a previously completed consumer, but this is not represented in the figure.

Notice that those nodes which lie at the front of the network must be executed before any nodes which lie after them can be reached. However, a node which has no others explicitly constrained to come after it and a low preference will exhibit the behaviour of being checked *every* time an action is picked. This is a useful feature for implementing *inter-suit* tactics (described in the next chapter), since they are typically executed at some indeterminate point when their preconditions become true.

Ensuring Continued Executability of Suppliers

One final point which is not addressed above is the way in which the resource analysis of a supplier may hide the fact that a resource may be utilised by the succeeding tactics in the node. For a Bridge example of this, consider the situation of Figure 8.24 on Page 231.

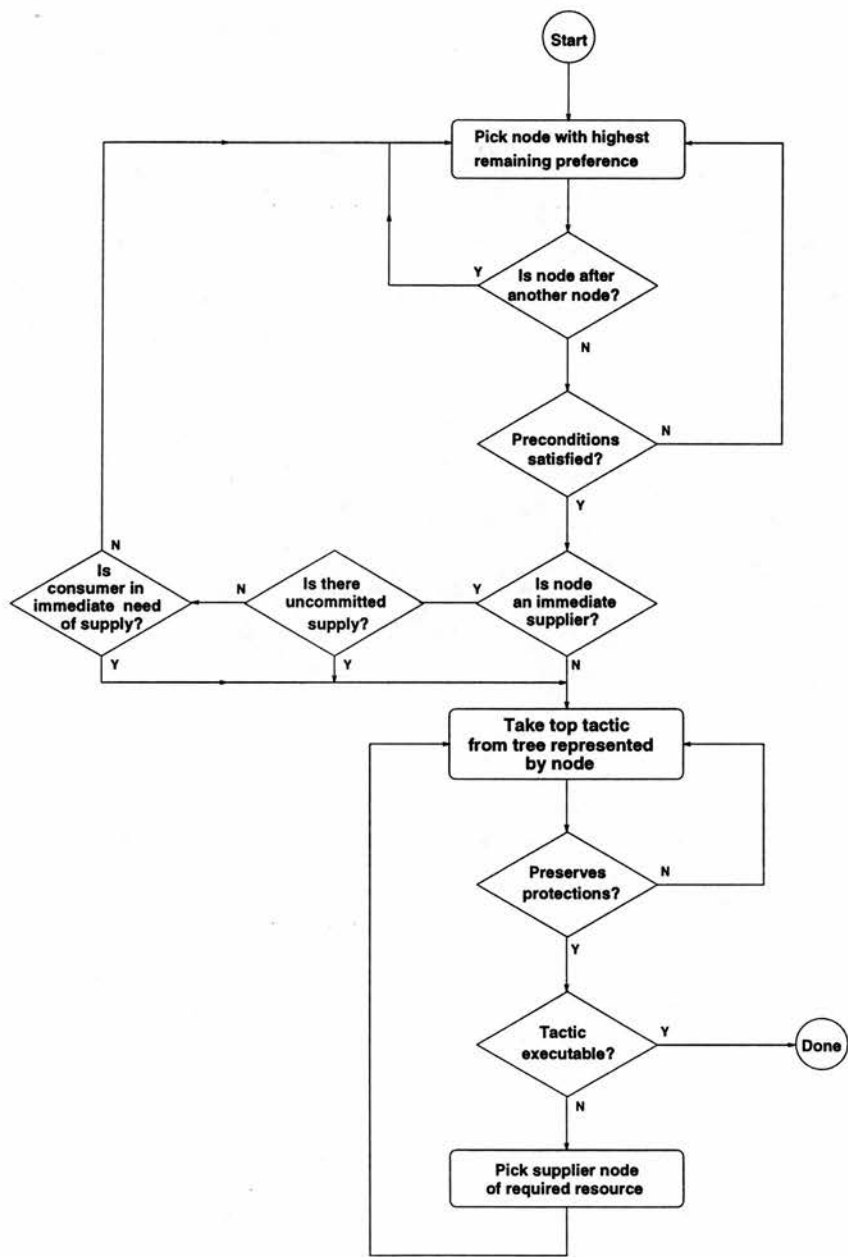


Figure 8.23: Using a network to order plan actions



Figure 8.24: A simple Bridge example

A plan to cash four tricks in this suit will generate one entry (resource) into the North hand. However, if this entry is immediately used by an action from another suit, a further entry will be required in order to cash the remaining tricks. We therefore include the extra constraint that when a `protect(Consumer, Supplier, Resource)` protection is in effect, any `Supplier` action that preserves the supply (and also does not waste resources by re-supplying the same resource again) *must* be executed before moving to the `Consumer`.

8.7 Summary

We have looked at how the single-suit lines of play produced by FINESSE can be interleaved together into a global plan. One important step that allowed this to be achieved was the identification of resource profiles for individual lines of play. These profiles were used as the basis for the refinement strategy in a partial-order plan-space system for describing the possible interleavings. Another important feature was the total ordering tractability refinement. We showed that this not only relieved us from having to consider *all* the possible orderings of a particular set of lines of play, but also side-stepped the problem of non-locality. Since this refinement removed the ability to compare different orderings of actions, we also described how other important features such as the desire to draw trumps could be represented explicitly using the notion of node preference.

Experimental results produced by FINESSE using this architecture can be found in Chapter 11. The implementation of the interleaver module provides support for the cliché-based approach of identifying formal structures describing the nature of a do-

main, and in particular demonstrates the feasibility of using resource-based reasoning to constrain the ordering of actions with disjunctive effects. The plans formed by the system are sound with respect to the resource-based profiles, in that if the notion of resources actually describes all the possible interactions between a set of lines of play and the interleaver succeeds in interleaving them, the lines of play will be executable together. However, since the system uses a heuristic ordering to assign suppliers to consumers and evaluates remaining resource supply on a conservative basis, it is possible that it will miss interleavings.

Chapter 9

Re-introducing Neglected Actions

No play carries more prestige than the squeeze. It is widely regarded as the expert's hallmark.

— TERENCE REESE AND DAVID BIRD
Bridge, the Modern Game

At the beginning of the previous chapter we identified three difficulties with the technique of solving a problem by breaking it down into smaller sub-problems which are solved independently. We then introduced an interleaving architecture which addressed the first of these: that of coping with possible dependencies when trying to combine together separate plans formed for the subgoals. Here, we address the second problem: the possibility that some domain actions will be overlooked when splitting up the overall task.

We examine this problem in the context of Bridge by considering a specific type of play called a *squeeze* play. We give examples of such plays that show how they can only be described by reasoning about multiple suits simultaneously. We show that although the basic FINESSE system will not discover such plays, it is possible to explicitly re-introduce them into the interleaving architecture as a special form of *inter-suit* tactic that describes possible plays involving cards from more than one suit. This is an important development since it suggests for the first time the feasibility of planning squeeze plays automatically. To date, the depth of lookahead required to identify possible squeeze situations has placed the task well beyond the reach of any computer system: the best that has been achieved is Nygate's PYTHON system (described in

Chapter 2), which can identify *existing* squeeze positions, but cannot determine the order in which cards should be played in order to *reach* a squeeze position. We are able, however, to show how a formalisation similar to that used by Nygate could be used to plan such plays automatically.

In §9.1 we describe the basics of squeeze play. §9.2 then describes an implemented example of FINESSE successfully incorporating a specific instantiation of a squeeze tactic into a global plan. This section also discusses some possibilities, not yet implemented, for extending this mechanism and making it more general, and suggests that our architecture is well-suited to supporting such reasoning. Finally, §9.3 examines the performance of other computer Bridge-playing software on such problems, and summarises the advantages of our approach.

9.1 The Simple Squeeze

According to [Reese & Bird 83, Page 185], ‘perhaps 90 per cent of all squeezes are variants of the so-called simple squeeze’. This is a play which occurs when declarer has the chance to make an extra trick in one of two suits. If one defender protects both of these suits, it may be possible to force him to discard a crucial card (*i.e.*, *squeeze* him) by cashing a winner in a third suit. If this can be achieved, declarer can then win an easy extra trick in the suit in which the defence had to relinquish its protection. To understand this better, consider the example of Figure 9.1 on the following page.

Playing in 7 No Trumps on the lead of the J♠, declarer has twelve top tricks and an easy thirteenth if either of the spades or the hearts break 3 – 3. If declarer simply tests each of these suits in turn, however, he will be disappointed. The suits do not break. If, on the other hand, he tests just one suit, say the hearts, and then cashes his winners, he will arrive at the position of Figure 9.2 on Page 236.

Now, when declarer plays the A♣, West is forced to make a choice in which he cannot win. Whatever card he throws away, declarer will make an extra trick, since if he plays the J♥ declarer will win an extra trick with the 8♥, and if he plays a spade, declarer will cash the A♠ and win an extra trick with the 7♠.

This play is an example of a *simple squeeze*. It has the following three major com-

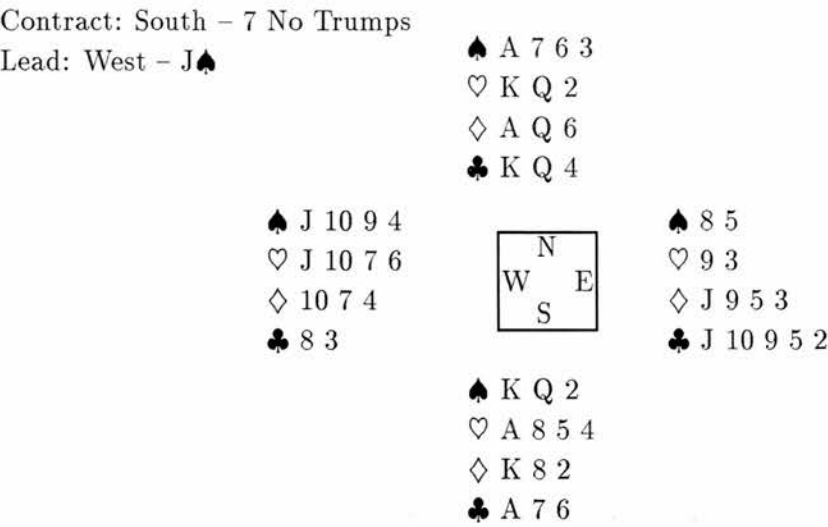


Figure 9.1: A Bridge deal allowing a simple squeeze ([Reese & Bird 83, Page 185])

ponents, which according to [Reese & Bird 83] are necessary for every simple squeeze, except for ‘a few rare and artificial positions’:

1. A *single menace*. A card which can be beaten by only one of the outstanding cards remaining in the same suit. The outstanding card which prevents the single menace from becoming a winner is termed the *guard*. In our example the single menace is the 8♥ and the defenders’ guard is the J♥.
2. A *two-card menace*. A winner accompanied by a card that can be beaten by one or two of the outstanding cards remaining in the same suit. In our example the two-card menace is the A7♠, and the guards for this are the 10♠ and the 9♠.
3. A *squeeze card*. A master that can force one of the defenders to abandon one of his guards (referred to as *unguarding*).

Notice that this simple squeeze is still successful if we switch East and West’s cards in the original hand, as in Figure 9.3 on Page 237.

East, like West in the original example, cannot guard both the spades and the hearts when the A♣ is played. This type of squeeze, which succeeds when either of the defenders holds the guards, is therefore known as an *automatic squeeze*. Compare this to the *positional* or *one-way squeeze* situation, of Figure 9.4 on Page 238.

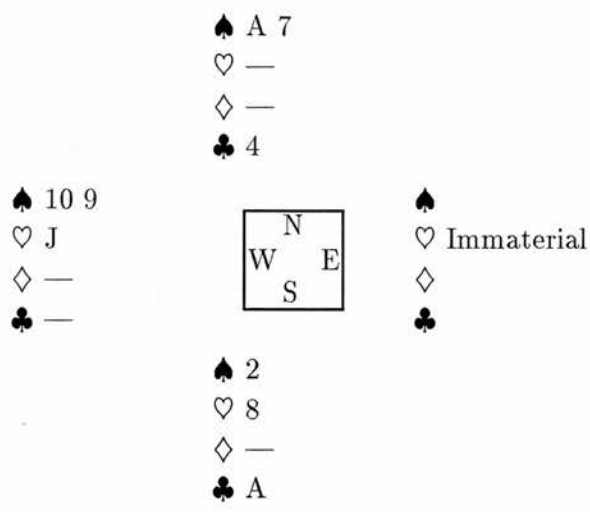


Figure 9.2: Situation when squeeze is actually applied

Here, the declarer again plays in 7 No Trumps and has 12 top tricks. He wins the Q♣ lead and tests the heart suit, finding that West holds four. Rather than relying on a 3-3 break in the spades he can now cash his top winners to reach the position of Figure 9.5 on Page 239.

When the squeeze card (10♦) is played, the declarer can determine which card to discard from dummy by looking at the discard selected by West. If West discards the J♥, declarer can afford to throw a low spade, and then win two tricks with the Q♠ and the 6♥. If West discards the J♠ or 10♠, declarer parts with the 6♥ and wins two tricks with the Q4♠. Unlike in the simple squeeze situation, however, this squeeze is not successful if the defenders’ cards are switched. If East holds the two guards, he will know which of the two to give up when he sees dummy’s play.

9.2 Re-introducing Squeeze Plays Into the Interleaver

To show the feasibility of incorporating squeeze plays in our interleaving architecture, we took our original example of Figure 9.1, designed a tactic to represent the (specific) situation, and added extra functionality to the interleaver algorithm so that it was able to cope with the tactic. Below, we describe the details of this extension.

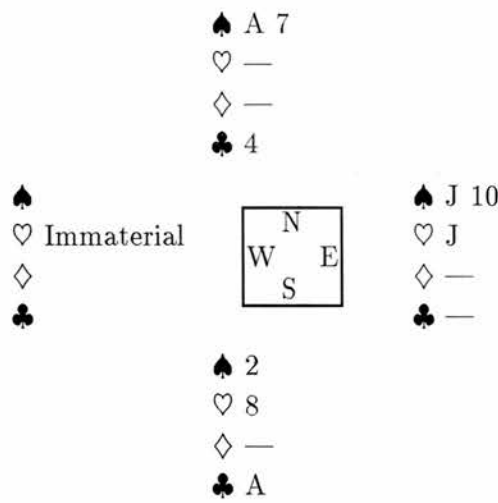


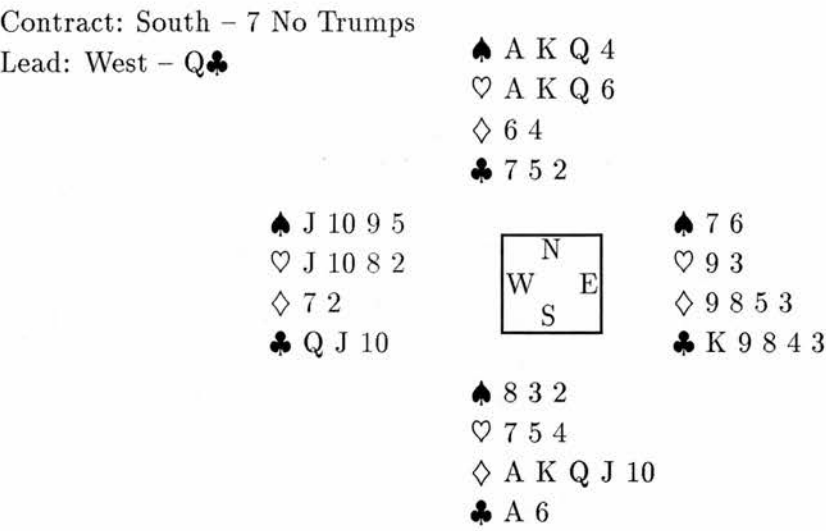
Figure 9.3: Squeeze situation with defenders' cards swapped

9.2.1 The Need For a Squeeze Tactic

We saw in the previous chapter that when the defenders do not follow suit, FINESSE's solution constructor function generally avoids considering the *actual* cards they might have played, and instead just represents their play as 'discarding'. This simplification enables the solution constructor function to restrict the line of play it constructs to a manageable size. Such an architecture, however, will not 'discover' squeeze plays, since their execution involves observing the actual identities of the cards thrown away by the defenders, deciding which threat cards have become established, and then proceeding to cash any extra tricks that have been created. Thus, in order to incorporate squeeze plays into the interleaving architecture, we design a special tactic which, when incorporated into the tree of tactics generated by the solution constructor function, pays special attention to the plays of some specific cards by the defence.

9.2.2 A Tailor-made Tactic

We represent the squeeze play of Figure 9.1 as an *inter-suit* tactic. Like the single-suit tactics we have already considered, this inter-suit tactic is a compound operator that represents multiple domain actions. However, rather than representing actions that form a single trick, our squeeze tactic represents the cards played over the *entire*



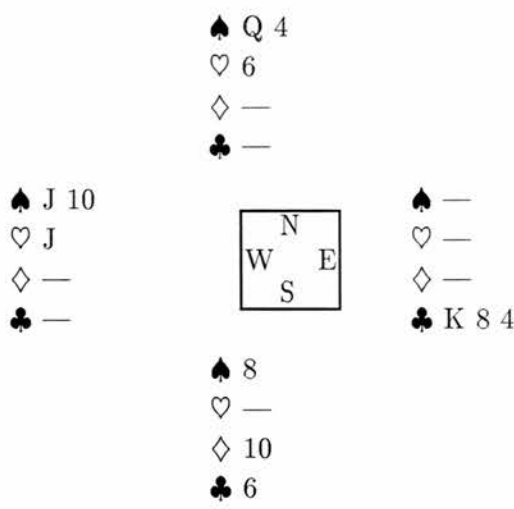


Figure 9.5: Situation when positional squeeze is actually applied

be added to the solution constructor function’s search space which explicitly caters for this possibility. In the following section we will show how the solution constructor function interprets this branch.

- **The interleaving constraints.** These are the constraints which must be obeyed by the solution constructor function if the squeeze tactic is to succeed. Each time a tactic is selected at a MAX node, these constraints will be checked. In this case, the constraints are that the A♣ and the A♠ must not be cashed.

Notice that there is no entry for the postconditions of the tactic. This is because, as with single-suit tactics, the possible MIN responses to a tactic (and hence the possible postconditions of a tactic) depend on the actual state of the world when the tactic is applied. We therefore calculate the postconditions of a squeeze tactic with the use of domain rules like those used to determine the effects of single-suit tactics.

Notice also that the actual card-plays that MAX would have to make in order to carry out the squeeze play are not specified. This may seem strange, but recall that the task of the solution constructor function is only to produce a profile of a given set of lines of play, inter-suit tactics and constraints. As long as the postconditions of each tactic can be calculated, this process does not require a knowledge of the tactics’ internal compositions. Where such knowledge *is* needed is in the plan execution module.

```

Tactic = [
    %name
        simple_squeeze(simple, automatic, s, h),

    %initial tactic
        cash(14,c),

    %interleaving preconditions
        [losers = 1, winners(c,d) = 1],

    %resource requirement
        entry(s,1)

    %branches generated
        [squeeze(automatic, s, h)]

    %interleaving constraints
        [protect(cash(14,c)), protect(cash(14,s))]
].

```

Figure 9.6: A squeeze tactic

However, for a squeeze play, this execution can be carried out entirely deterministically, basing each choice of play purely on the cards played by the defenders on the previous trick. [Nygate 84] has written on this subject, and demonstrated that a simple set of rules is sufficient to guide the execution of a squeeze play, given the particular type of squeeze being attempted and a specification of the threat cards. We see no theoretical difficulties with the eventual incorporation of such rules into a plan execution module for FINESSE.

9.2.3 Carrying Out The Interleaving

In our example, a successfully interleaved simple squeeze tactic can be regarded as a single compound entity covering three rounds of play and gaining three tricks for the declarer. However, when attempting to produce an interleaving it is also important to consider the possibility that the attempted squeeze may fail. Figure 9.7 depicts FINESSE's solution to this problem.

When the solution constructor function reaches a state in which the preconditions of

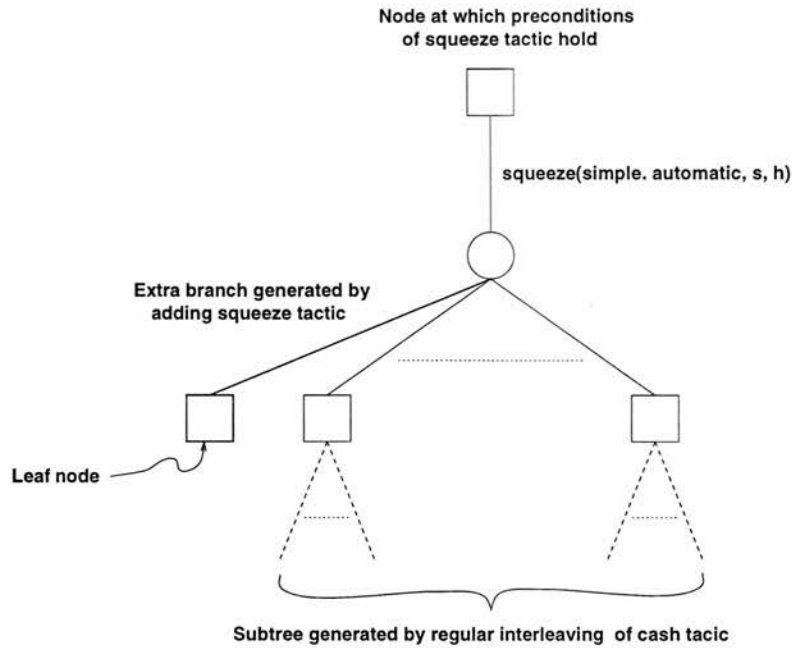


Figure 9.7: The interleaving of a squeeze play

the squeeze tactic succeed, it first searches the space that would result if the play of the initial squeeze tactic (in our example, the cashing of the $A\clubsuit$) were incorporated as a normal tactic. As we have already pointed out, this process will *not* realise that extra tricks are available, as it pays no attention to the particular cards played by the defenders when they are discarding. Nevertheless, this subtree is analysed in the normal way, and some profile P , produced. Once the analysis of this sub-tree is complete, an extra branch representing the three rounds of defenders' plays which *will* lead to an extra trick is added to the initial MIN node. This branch is then examined to determine the worlds, C , under which it would succeed (which are exactly the worlds in which the guards are held by a single defender) and the number of tricks, T , this would produce.

The way to view these branches is that the squeeze branch will be followed in any world in which the squeeze succeeds, and the remaining branches followed otherwise. The worlds in which the squeeze branch can be followed therefore subsume the possible worlds under which the remaining branches can be followed. Thus, the pair $T \triangleright C$ is added to the original profile, P , after carrying out subsumption.

9.2.4 Constructing a General Method for Simple Squeezes

Although our tactic of Figure 9.6 is tailored to suit the particular game we are considering, it serves the purpose of demonstrating that squeeze plays can be incorporated into FINESSE's interleaving architecture and demonstrates the general feasibility of using inter-suit tactics. The task of determining which particular instantiation of a squeeze tactic should be applied in any given situation would properly be the job of a squeeze method, or methods. FINESSE currently has no such methods, which would need to be able to examine a deal and suggest what type of squeeze play (if any) may be applicable. However, we suggest that for the particular problem of squeeze plays, the major obstacle to their incorporation in an automated Bridge system lies not in this area, but in the task of actually projecting the play forward far enough to be able to test such hypotheses in reasonable time. We have shown above that at least in principle FINESSE can tackle this problem.

It is also a curious fact that although squeeze plays are a relatively advanced concept in the game of Bridge, they are described by a reasonably clear theory. Certainly, there is extensive literature on how to recognise squeeze plays. [Love 59], for example, identifies explicit conditions that must hold for a squeeze to be applicable, and demonstrates how to check for these conditions. This was the work used by [Nygate 84] to design the PYTHON expert system for recognising squeeze plays. PYTHON, however, was only able to recognise when squeeze situations were *immediately* applicable. We anticipate that the adaptation of theories like those of [Love 59] will enable FINESSE to actually plan for the play of such actions from the start of the game.

9.3 Performance & Discussion

Despite having to project play through to the end of the game, FINESSE manages to interleave the squeeze play in our example without being unacceptably slow. It requires approximately twenty seconds to solve the problem, in which time it plans and interprets the individual suits, interleaves a line of play without the squeeze tactic (producing 13 tricks if either the spades or the hearts split 3-3), and interleaves a line of play including the simple squeeze. As we have pointed out already, once FINESSE's

planning is over and the best plan has been selected, execution of the squeeze play should require no extra thinking time, since it can be governed by a simple set of rules.

To demonstrate that the planning of squeeze plays is a hard problem we gave our example deal to two commercial Bridge systems, and found that both of them were unable to find the correct play.

First, we tested Micro Bridge Companion (see §A.15), which contains the ‘Bridge Baron’ playing algorithm that has won the Bridge contest at two Computer Games Olympiads. We set the program to its top difficulty setting, in both duplicate and rubber Bridge modes, but each time it followed the same course of action, winning the J♠ lead with the K♠, then cashing three rounds of hearts followed by the Q♠ and the A♠. Typical thinking time for each card play was of the order of 3 or 4 seconds. MBC also had problems with the bidding, stopping in 3 No Trumps. We suspected that this may have been the reason for its failure to look for a squeeze during the play phase, but the program chose exactly the same line of play when the contract was manually set to 7 No Trumps.

Our second test program was Grand Slam Bridge (see §A.11). This system was able to bid to 6 No Trumps, but during the play phase it then appeared to proceed cash its winners in a pseudo-random fashion. In five attempts, it once got to within one trick of a squeeze situation before erroneously cashing the winner from its two-card menace.

Thus, FINESSE was the only system to recognise the possibility of a squeeze, and it was also able to incorporate this play into its plan from the start of the game. Of course, FINESSE was deliberately given the specific tactic necessary to cope with the situation, so its success is maybe not surprising. However, the comparative results highlight the difficulty of planning squeeze plays, and we have shown that the architecture of FINESSE at least has the potential to support the automatic recognition of squeeze situations.

FINESSE also has the potential to explain *why* it is playing the tricks in a given order, since it reasons about its actions at a level that would make sense to a human Bridge player. Further, it has the necessary information to explain the chances of success of its actions within the terms forming its uncertainty language. Currently, FINESSE can

only generate probabilities from its uncertainty terms that describe complete games (although we show in Appendix B how textual explanations can be produced for \mathcal{C} -conjunctions describing single suits). However, the actual term produced by FINESSE to describe the chance of winning 13 tricks in our simple squeeze example is:

```

[[east_low(1111111)], [east_low(0001000)], [east_low(1111111)], [east_low(1111111)]]
∨ [[east_low(0001000)], [east_low(1110111)], [east_low(1111111)], [east_low(1111111)]]
∨ [[east_low(0000100)], [east_low(0000100)], [east_low(1111111)], [east_low(1111111)]]
∨ [[east_low(0010000)], [east_low(0010000)], [east_low(1111111)], [east_low(1111111)]]

```

Each disjunct in this term represents disjoint sets of distributions, and contains one `east_low` term describing the low cards in each of the suits spades, hearts, diamonds and clubs, respectively (there are no critical cards in this situation). From top to bottom, these disjuncts represent the following situations: ‘The hearts are split 3-3 (and maybe the spades are also split 3-3)’, ‘The spades are split 3-3, but not the hearts’, ‘West holds four hearts and four spades’, ‘East holds four hearts and four spades’. The first two of these are the situations under which simply cashing the top cards would produce 13 tricks. The second two describe the extra distributions under which the squeeze tactic would succeed.

9.4 Summary

We presented the simple squeeze play as a Bridge example of how some optimal solutions may be overlooked when splitting a task into a number of sub-problems. We showed that correctly understanding squeezes required reasoning about the situations in more than one suit simultaneously.

We demonstrated that although the single-suit decomposition in FINESSE results in such plays being overlooked, it is possible for them to be re-introduced into the interleaver algorithm via the use of a specially-designed inter-suit tactic. We gave an example of an actual interleaving of such a tactic to demonstrate the feasibility of this approach and also showed that other leading Bridge programs were unable to identify this squeeze situation.

Finally, we speculated that the specification of squeeze tactics as methods would allow

FINESSE to identify potential squeeze situations automatically, and also that the representation used by the system could be utilised to generate high-level explanations of its actions in such situations.

Chapter 10

Overall Architecture

He became aware, very slowly, very gradually, that he held some impossibly complex model of the contest in his head, unknowably dense, multifariously planed.

— IAIN M BANKS
The Player of Games

You're about to get what you've always wanted — but is it as good as you originally thought? It's never too late for a change of plan.

— Author's horoscope
TV Quick magazine, May 16 1992

In the previous chapters we have discussed the problems involved in producing a Bridge planning system, studying separately the issues of:

- generating the legal play sequences for single-suit Bridge play problems (Chapter 5),
- selecting the most promising lines of play from amongst these sequences (Chapter 6 & Chapter 7),
- identifying the constraints which are necessary in order to combine more than one such line of play into a global plan (Chapter 8), and
- re-introducing neglected actions with inter-suit tactics (Chapter 9).

In this chapter, we draw together these developments into a single architecture capable of producing plans for the game of Bridge. This architecture addresses the final problem we identified at the beginning of Chapter 8: that when solving a given problem

by splitting it into subgoals, the best solution may not simply be composed of a combination of the best solutions to these subgoals, but could instead involve sub-optimal solutions to some or all of them. Although FINESSE currently only uses this architecture to produce plans for No Trumps contracts, the potential to deal with situations involving a trump suit is retained by the ability to incorporate inter-suit tactics.

We begin in §10.1 by outlining a simple planning framework. We then show how the plan *profiles* described in previous chapters can be utilised within this framework; in §10.2, we define a partial ordering on probability profiles, and in §10.3 we look at how profiles can be used to make efficiency improvements. §10.4 then describes the system architecture incorporating these enhancements, and §10.5 gives a summary.

10.1 A Simple Planning Loop

We begin by giving a simple planning loop which draws together the developments we have described in the previous chapters (see Figure 10.1). This loop returns a set of *plan objects*, which for any global plan we define as being:

1. the lines of play to be interleaved,
2. the inter-suit tactics introduced by the inter-suit methods,
3. the ordering constraints introduced by the interleaver, and
4. a global profile of the interleaving, produced by the solution constructor function.

An important feature of this planning loop is the step which ‘selects lines of play’. Currently, FINESSE tackles this problem by initially counting the *top tricks*. These are the tricks which can be won by simply cashing winners — something which FINESSE models by analysing the minimax trees for each suit with a restricted version of the interpreter algorithm that only considers cash tactics. Interleaving the lines of play produced by this restricted interpreter algorithm results in the ‘worst best’ global plan of cashing all the masters. This plan can then be improved upon either by adding inter-suit tactics, or by improving the line of play in one or more suits. Note that the practice of beginning by counting the top tricks is common among human Bridge

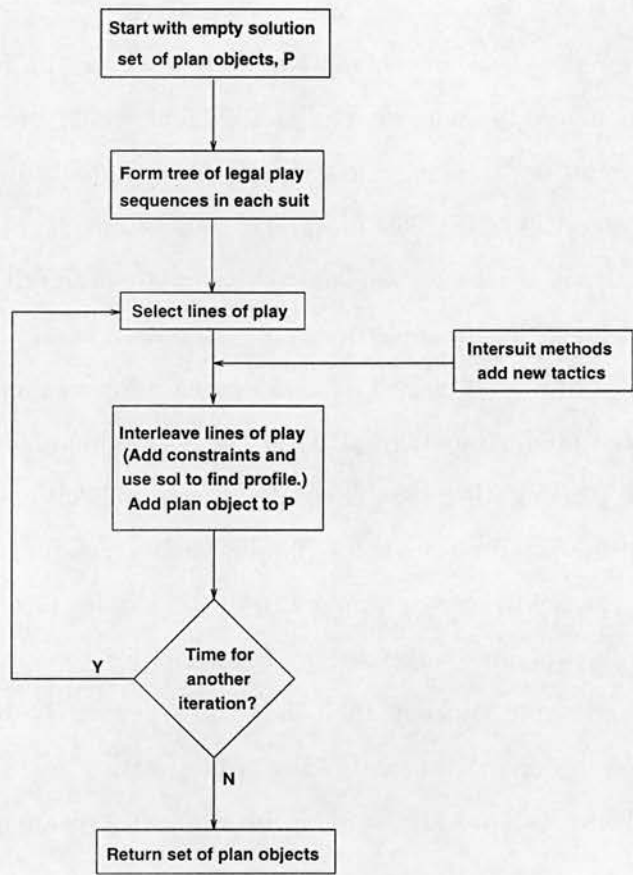


Figure 10.1: Simple planning loop for an anytime architecture

players, at least when playing in No Trumps, and is often explicitly recommended in Bridge books, *e.g.*, [Reese & Dormer 91, Page 29].

The current version of FINESSE uses the very simple control structure of trying only the ‘best’ line of play (as identified by the interpreter algorithm) or the top tricks line of play in each suit. Simple backtracking is used to try all sixteen possible combinations of these lines of play (plus inter-suit tactics), if planning time permits. Ideally, information from the failure of the interleaving step, or from the result of interpreter step would also be utilised in the control loop. This would allow lines of play different from the ‘best’ in each suit to be identified by altering the definition of the partial ordering on payoff profiles introduced in §8.2.2. Such a modification would enable goals such as ‘developing entries in a suit’ or ‘keeping one defender away from the lead’ to be pursued. However, even without such a control structure, FINESSE is able to solve

difficult problems, as we shall see in Chapter 11.

The reason for the check on planning time is that any Bridge program called upon to actually compete in some kind of tournament would presumably be under some kind of constraint on ‘thinking time’ within which to make its moves. This constraint calls for an algorithm that can be interrupted at any point during computation to return a result, in a manner similar to the *anytime* algorithms for time-dependent planning developed by [Dean & Boddy 88, Boddy & Dean 89]. By starting with the top tricks lines of play in each suit and gradually upgrading them, FINESSE is more likely to at least produce *some* result after a small amount of planning time, whereas the alternative of starting with ‘best’ lines of play for each suit and then backtracking, say, may use a lot of planning time before finding *any* successful interleavings. Fortunately for FINESSE, the key to success in many Bridge situations typically lies in concentrating on just one or two suits whilst using the others for communication, or to stop the defenders from taking tricks in their long suits. Thus, the best overall plan is often a combination of one or two ‘best’ lines of play with the cashing of top tricks in the other suits, which can quickly be found by FINESSE’s planning loop.

The overall result of this planning loop is generally a *set* of plan objects. Before any actions can be carried out, then, it is necessary to select a single member of this set. In the following section, we examine how the probability profiles produced by the solution constructor function can be used to achieve this.

10.2 A Partial Ordering on Profiles

It is often difficult to make an absolute judgement about whether one profile represents a ‘more promising plan’ than another. For instance, consider the profiles depicted in Figure 10.2 (recall that any profile contains a set of non-zero probabilities, p_i , of making *exactly* T_i tricks.)

On the basis of an expected value computation, we would prefer the first profile. However, this is by no means the only criterion we may choose. For example, we may be interested in restricting our losses to a minimum, in which case the second profile would be the most promising. Other possible criteria might be to prefer the

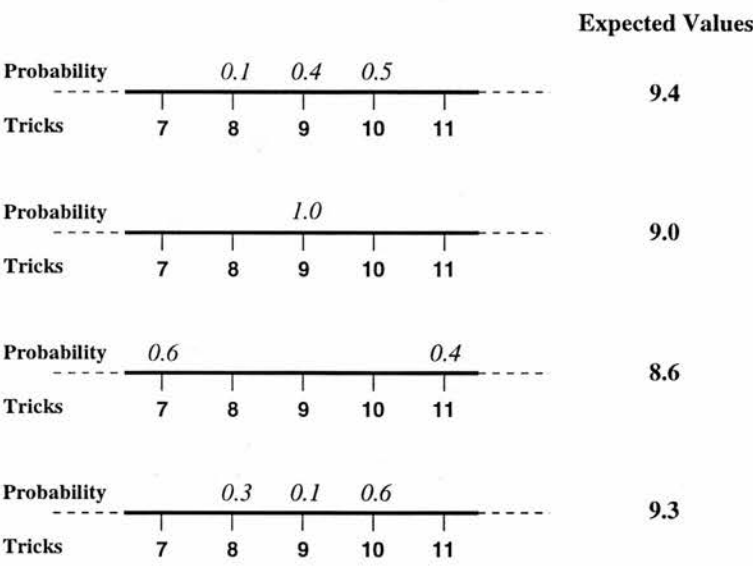


Figure 10.2: Schematic representation of four plan profiles

profile with the highest chance of making the largest number of tricks (the third, in Figure 10.2), or the profile with the largest chance of making at least a certain number of tricks (the last, if we look for 10 tricks).

Each of these criteria forms a feasible basis for comparing profiles, and yet since each would lead to a different profile being preferred from the set in Figure 10.2, we can see that they are mutually incompatible. In FINESSE, we wanted to have the ability to change the selection criterion used by the system. However, we also wanted to be able to identify profiles that were clearly inferior under *any* ‘reasonable’ criterion. Rather than the *total* orderings imposed by the criteria discussed above, then, FINESSE uses a *partial* order, to enable a *set* of solutions to be retained as ‘best’ candidates. The first step in defining this partial order is the introduction of the following *shift* operation:

Definition 10.1 (Shift Operation on Profiles) To *shift* a given profile $[T_1 \triangleright p_1, T_2 \triangleright p_2, \dots, T_l \triangleright p_l]$ (to the right), for some $1 \leq i \leq l$ a value $0 < \rho \leq p_i$ is selected and the profile rewritten as $[T_1 \triangleright p_1, \dots, T_i \triangleright (p_i - \rho), T_{i+1} \triangleright (p_{i+1} + \rho), \dots, T_l \triangleright p_l]$ if T_{i+1} is one greater than T_i , or $[T_1 \triangleright p_1, \dots, T_i \triangleright (p_i - \rho), (T_i + 1) \triangleright \rho, \dots, T_l \triangleright p_l]$ otherwise.

Effectively, modifying a profile in this way increases by one trick our anticipated pay-

off for some of the conditions under which T_i tricks were previously being expected. Clearly, any ‘reasonable’ selection criterion should regard a profile modified in this way as better than the original. Based on this operation, then, we can define a binary relation \geq on profiles, similar to the *weak dominance* of classical game theory [Fudenberg & Tirole 95], as follows:

Definition 10.2 *For any two profiles ϕ_1 and ϕ_2 , $\phi_1 \geq \phi_2$ if and only if by zero or more applications of the shift operation, ϕ_2 can be made identical to ϕ_1 .*

It is a simple matter to check that this relation defines a partial ordering on the set of all possible profiles (*i.e.*, that it satisfies the reflexive, antisymmetric, and transitive laws). This partial ordering allows us to retain a set of potential solutions, different members of which may be regarded as the ‘best’ by criteria that impose a total ordering. Let us define such a set as follows:

Definition 10.3 (Irredundancy) *A set of profiles, Φ , is irredundant if it contains no two elements ϕ_1 and ϕ_2 such that $\phi_1 \geq \phi_2$.*

Returning to the profiles of Figure 10.2 it is a simple matter to check that none of these profiles can be transformed into any of the others, and that the set of profiles is therefore irredundant. Adding the further profile

$$[7 \triangleright 0.6, 9 \triangleright 0.4]$$

to the set, however, would result in it losing the property of irredundancy as this profile can be transformed into *any* of the profiles of Figure 10.2 by applications of the shift operator. The profile

$$[8 \triangleright 0.1, 9 \triangleright 0.1, 10 \triangleright 0.8] ,$$

on the other hand, cannot be transformed into any of the profiles in the figure. Rather, the first and last profiles can be transformed into this one. Therefore, if this profile is added to the set, the first and last profiles would have to be removed to maintain irredundancy.

10.2.1 A Modified Planning Loop

We can make use of the notion of irredundant profile sets to decide which set of plan objects to return as possible solutions to a given problem. Given a planning loop which has already generated a series of plan objects, the profiles of which form an irredundant set Φ , a further candidate plan object with profile ϕ_1 is retained as a possible ‘best’ solution if there is no element ϕ_2 of Φ such that $\phi_2 \geq \phi_1$. If this is the case, the new plan object is added to the original set, and the new set of profiles $\Phi \cup \phi_1$ is made irredundant by removing any plan objects which have profiles, ϕ_2 , for which $\phi_1 \geq \phi_2$. A planning loop based on this principle is shown in Figure 10.3.

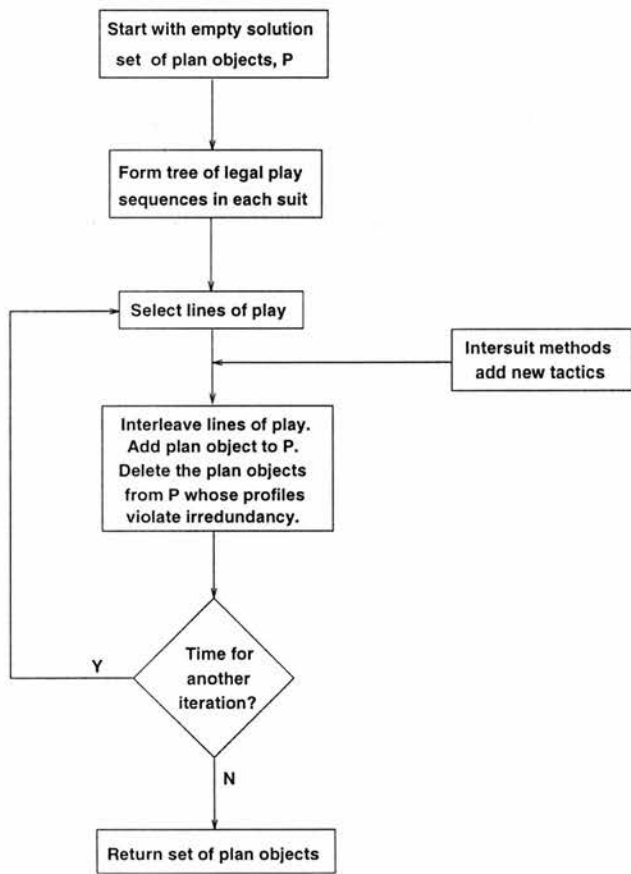


Figure 10.3: Simple planning loop for an any-time architecture

This is clearly a useful way to define the ‘solution’ to a Bridge problem, since the type of behaviour demonstrated by the system can be modified by changing the (total ordering)

criterion by which the ‘best’ profile is selected from amongst the final, irredundant set. For example, an aggressive approach of always attempting to maximise overtricks or a cautious approach of always optimising the chance of making the contract can both result from the same planning system. The actual choice of criterion for selecting the plans will probably depend on the type of game being played. For example in *rubber* Bridge, the most important goal is to make a specified contract. In *duplicate* contests (where many teams of two players play the *same* cards), it is typically more profitable to attempt to make overtricks where possible. It is even possible to envisage a higher-level routine which switches between a variety of total ordering criteria depending on the current state of the play (*e.g.*, the current scores, or the *vulnerability* of each side).

FINESSE currently implements four styles of play which it terms **expected**, **conservative**, **aggressive**, and **make_contract**. These correspond to the four criteria (discussed at the beginning of §10.2) of expected value computation, the smallest chance of winning the lowest number of tricks, the highest chance of winning the largest number of tricks, and the highest chance of making at least enough tricks to make a contract. However, FINESSE has no mechanism for choosing between these styles; this choice must currently be made by the user.

In addition to supplying a useful way to compare profiles we can use the above developments on analysing profiles to make an efficiency improvement to the planning loop. This is discussed in the next section.

10.3 Forming ‘Best-Case’ Profiles

The interleaving step of our planning loop is expensive, since it involves a call to the solution constructor function. However, in previous chapters we have already seen that as well as being able to produce global profiles, FINESSE is also capable of describing the results of single-suit lines of play with probability profiles. We now examine how such profiles can be used to produce an *estimate* of the profile that would result from the interleaving of some given lines of play. We then show that this estimate can be used to ‘prune’ unpromising interleavings.

10.3.1 A Simple Approach

Let us say that we are considering interleaving four arbitrary lines of play. FINESSE's interpreter algorithm can be used to produce four probability profiles of these lines of play, of the form

$$L_i = [T_{i1} \triangleright p_{i1}, T_{i2} \triangleright p_{i2}, T_{i3} \triangleright p_{i3}, \dots] , \quad (10.1)$$

where p_{ij} is the probability of obtaining T_{ij} tricks by following the given line of play for suit i ($i = 1, 2, 3, 4$).

The simplest conceivable way of using these profiles to estimate the number of tricks that would be produced when executing all four lines of play together would be to combine the p_{ij} as follows:

$$\Pr(T \text{ tricks in total}) = \sum_{i=1}^{l_1} \sum_{j=1}^{l_2} \sum_{k=1}^{l_3} \sum_{l=1}^{l_4} p_{1i} \times p_{2j} \times p_{3k} \times p_{4l} , \quad (10.2)$$

$$T_{1i} + T_{2j} + T_{3k} + T_{4l} = T$$

where l_i is the length of the list L_i . However, the probabilities we are dealing with here are not independent. For example, there could be an i, j, k , and l for which they each describe the probability of one of the defenders being void — feasible individually but impossible in conjunction. Fortunately, the qualitative nature of the uncertainty representation language used by FINESSE allows us to reason about such possible dependencies by replacing the probabilities in (10.1) by the terms, C_{ij} , (disjunctions of \mathcal{C} -conjunctions) from which they were originally derived:

$$L_i = [T_{i1} \triangleright C_{i1}, T_{i2} \triangleright C_{i2}, T_{i3} \triangleright C_{i3}, \dots] . \quad (10.3)$$

With this representation, we can now examine the *actual* distributions represented by the uncertainty language, and determine a probability by considering only those distributions which are valid.

$$\Pr(T \text{ tricks in total}) = \sum_{i=1}^{l_1} \sum_{j=1}^{l_2} \sum_{k=1}^{l_3} \sum_{l=1}^{l_4} \text{prob}(C_{1i} \wedge C_{2j} \wedge C_{3k} \wedge C_{4l}) \quad (10.4)$$

$$T_{1i} + T_{2j} + T_{3k} + T_{4l} = T$$

To implement the $\text{prob}(\dots)$ notation here, we return to the shape-list algorithm from §7.4.6. Recall that this algorithm constructs lists of 14 entries in which the number of distributions where East holds n cards is represented by the $(n-1)$ th element of the list. The algorithm for producing a shape-list for an isolated term, C_1 , first requires the initialisation of the variable `list_in` to the list $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ (to represent the fact that East has not yet been assigned any cards), and then processes C_1 to produce an answer `list_out`. To use this algorithm to produce a shape-list for a conjunction of terms, $C_1 \wedge C_2 \wedge \dots$, we simply use the `list_out` resulting from the processing of C_1 as the `list_in` for the processing of C_2 , and so on. If there are N cards outstanding, and we know that East holds N' of them, the probability that the conditions described by the terms $C_1 \wedge C_2 \wedge \dots$ hold is found by taking the $(N'+1)$ th element of the final shape-list, and dividing this by the total number of ways in which to distribute the remaining cards between the two defenders, which is $N!/N'!(N-N')!$. We thus have a way, based on the shape-list algorithm and the profiles of (10.3) of respecting the inherent dependencies when combining together some lines of play.

10.3.2 Don't Count Your Chickens...

Unfortunately, (10.4) does not capture all the complications that are present in Bridge. For instance, it may occur that the promising potential shown by a line of play being interleaved is the result of the strength deriving from a particularly long suit in either declarer's or dummy's hand. When such a plan is executed, the declarer will have to find *discards* from the hand which is short in the suit, since one card *must* be contributed from each hand to each trick. This may then affect the number of tricks available in the other suits. As an example, consider the deal of Figure 10.4.

According to FINESSE, the best possible plans for each of the suits in this deal offer a maximum of three tricks in the spade suit (if the K_{\spadesuit} is singleton), two tricks in the heart suit, five tricks in the diamond suit, and four tricks in the club suit. Since the

♠ Q J 5
♥ K 8
♦ 7 6 2
♣ K J 10 9 6



♠ A 6
♥ A 4 2
♦ A K Q J 10
♣ 8 4 3

Figure 10.4: An example Bridge deal

conditions under which each of these numbers of tricks may be successfully returned are not mutually exclusive, (10.4) will therefore assign a non-zero possibility to the chance of obtaining 14 tricks!

Although at first the solution may sound *ad hoc*, this problem can be easily solved by simply restricting the number of tricks to be at most the same as the maximum number of cards held in the declarer’s or dummy’s hand. For instance, consider the North hand in Figure 10.4. When hoping to win three tricks in the spade suit and two tricks in the heart suit, all North’s cards in these suits will have to be played. However, in the diamond suit North only has three cards, whereas five tricks are planned. Two cards from another suit will therefore have to be thrown away. In the club suit there is one ‘spare’ card, since the best possible result is four tricks and North holds five cards. However, this still means that one card, which had previously been assessed as a winner must be thrown away.

In the general case, let us number the suits $i = 1, 2, 3, 4$ and say that we are examining the possible worlds in which we expect our lines of play to achieve T_i tricks in each suit ($i = 1, 2, 3, 4$). If we use the function $numcards(P, i)$ to represent the number of cards held by the player P in suit i , then for suit i the number of ‘spare’ cards — that is, those not committed to winning a trick — in North’s hand is given by

$numcards(north, 1) - T_1$. The total number of spare cards in North's hand is therefore

$$\begin{aligned} & \sum_{i=1}^4 (numcards(north, i) - T_i) \\ &= \sum_{i=1}^4 numcards(north, i) - \sum_{i=1}^4 T_i \\ &= N - \sum_{i=1}^4 T_i, \end{aligned}$$

where N is the total number of cards in North's hand. Now if $N - \sum_{i=1}^4 T_i$ is less than 0 it describes the number of winners which must be lost by throwing North's contribution to the winning trick away as a spare card. *I.e.*, if $\sum_{i=1}^4 T_i > N$, then the number of tricks available is

$$\sum_{i=1}^4 T_i + (N - \sum_{i=1}^4 T_i) = N.$$

The same argument holds for South's cards, so the maximum number of tricks that can be taken by the declarer is the maximum length of the North or the South hand. With this restriction, we can now rewrite (10.4) as

$$\Pr(T \text{ tricks in total}) = \sum_{i=1}^{l_1} \sum_{j=1}^{l_2} \sum_{k=1}^{l_3} \sum_{l=1}^{l_4} \text{prob}(C_{1i} \wedge C_{2j} \wedge C_{3k} \wedge C_{4l}), \quad (10.5)$$

$\min(T_{1i} + T_{2j} + T_{3k} + T_{4l}, M) = T$

where M is the maximum of the number of cards held in the North hand and the South hand. For the situation of Figure 10.4, (10.5) reduces the maximum possible number of tricks from 14 to 13. However, even this still overlooks one issue: the problem of losers.

10.3.3 Losers

It is very common that, in order for the declarer to take his tricks in any one suit, some tricks will inevitably have to be conceded to the defence. For instance, in our example deal of Figure 10.4 it is inconceivable that the declarer could make four tricks

in the club suit without conceding at least one to the defenders, who hold the Ace. Thus, as well as requiring knowledge of the actual conditions under which each number of tricks is won rather than just the bare probabilities, it is also necessary to know how many losers may be incurred, and the conditions under which these losses will be made. Fortunately, this information is also readily available from FINESSE's interpreter algorithm, which can be simply extended to generate modified profiles of the following form

$$L_i = [(T_{i1}, U_{i1}) \triangleright C_{i1}, (T_{i2}, U_{i2}) \triangleright C_{i2}, (T_{i3}, U_{i3}) \triangleright C_{i3}, \dots], \quad (10.6)$$

where U_{ij} represents the number of times that the defence will be given the lead when making T_{ij} tricks, and all the pairs (T_{ij}, U_{ij}) are distinct. This profile is produced by processing the original profiles containing payoff 8-tuples, π_{ij} , to collect together the possible worlds C_{ij} for which $tricks(\pi_{ij})$ and $concede(\pi_{ij})$ are identical (see Definition 8.2 on Page 202). The U_{ij} are therefore a lower bound on the number of losers, as the lines of play never allow the defenders to control the lead, and therefore never allow for the possibility of the defenders using such control to cash further tricks in their strong suits. Using this information, then, we can finally construct a profile for the interleaving of four lines of play using the formula

$$\Pr(T \text{ tricks in total}) = \sum_{i=1}^{l_1} \sum_{j=1}^{l_2} \sum_{k=1}^{l_3} \sum_{l=1}^{l_4} \text{prob}(C_{1i} \wedge C_{2j} \wedge C_{3k} \wedge C_{4l}) \cdot \quad (10.7)$$

$$\min(T_{1i} + T_{2j} + T_{3k} + T_{4l}, M - U_{1i} - U_{2j} - U_{3k} - U_{4l}) = T$$

In the deal of Figure 10.4, any attempt to attack the club suit will always incur one loser,¹ so the maximum number of tricks will never rise above 12. In general, the profile produced by (10.7) will represent an ‘upper bound’ (with relation to the shift operator) on the *best-case* profile that can result from the interleaving of the lines of play, since no attempt has been made to reason about the conflicts which may have

¹ Note that the (T_{ij}, U_{ij}) pairs are produced under an assumption of best defence. For example, given a choice between the declarer winning four tricks whilst losing none or winning four tricks whilst losing one, the defenders are assumed to choose the latter. In this particular example deal, then, although it is possible to win four tricks in the club suit without the defence taking any tricks, the best defence assumption has the effect of ignoring all the defenders' lines of play that do not include the cashing of the Ace.

arisen due to the false assumption of independence when decomposing the original task into four sub-goals. These dependencies may conspire to reduce the chances of making certain numbers of tricks, for example due to a lack of entries or to the defenders establishing their own tricks before the declarer can cash his winners.

10.4 An Improved Overall Architecture

The practical use of best-case profiles in FINESSE is in improving the efficiency of the planning loop. For, if the loop has already generated a set of plan objects whose profiles form an irredundant set Φ , we can use the best-case profile, ϕ_1 , of a new set of lines of play to disregard unpromising combinations *before* attempting to interleave them. This is done by checking that there are no elements, ϕ_2 , of Φ for which $\phi_2 \geq \phi_1$, and only calling the interleaver algorithm if this is found to be true.

Figure 10.5 shows an improved planning loop which incorporates this step. Clearly, if the cost of generating best-case profiles is cheap compared to the cost of interleaving a set of plans, this will be a useful efficiency measure. In FINESSE, the best-case profiles are very simply produced, as all the information required is present in the lines of play produced by the interpreter algorithm.

We can now finally come full circle and return to the figure depicting the system architecture first presented in Chapter 1. All the stages in this architecture have now been described, with the exception of plan execution portion of the play module, which has yet to be implemented. We therefore repeat the system architecture in Figure 10.6, expanding slightly on a few details. Notice that the entire system is encapsulated in an interface module. We have seen some screen captures from this interface already in Chapter 5. Further examples of the system's appearance can be found in Chapter 11 and Appendix C.

10.5 Summary

We have presented the overall architecture of the FINESSE system, drawing together the developments of the previous chapters into a single, modular, anytime planning

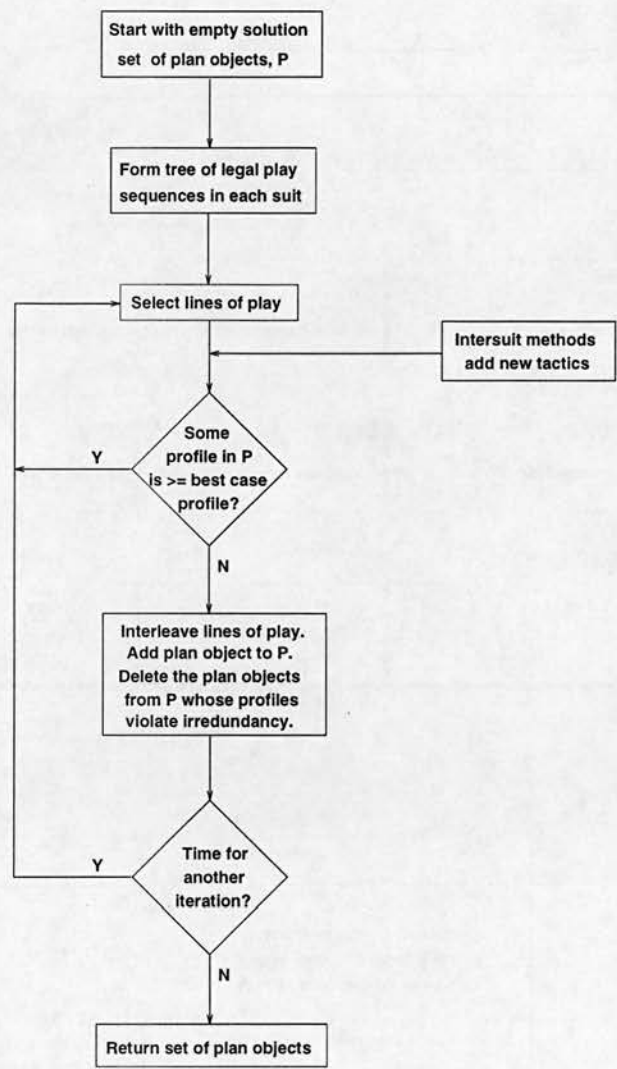


Figure 10.5: Planning loop modified for efficiency

system. In particular, we discussed the notion of global profiles, introducing a partial ordering for comparing such objects and defining the notion of an irredundant set of profiles. We noted how different styles of play could be produced by using different criteria to select between the profiles in such a set.

We also showed how a best-case profile for the interleaving of four lines of play could be produced from independent analyses of the individual lines of play. This development was then utilised in the overall planning loop to improve the system's performance.

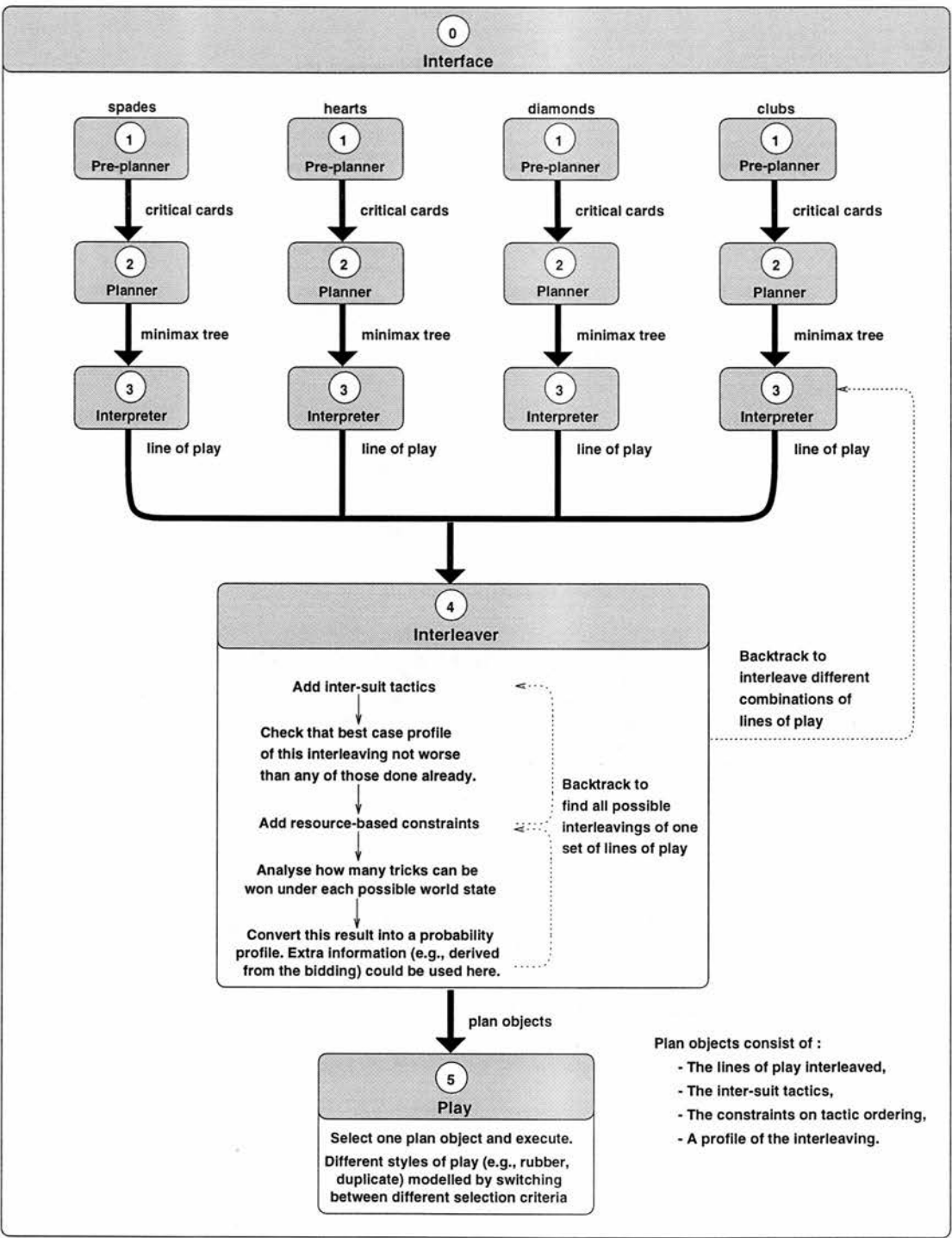


Figure 10.6: The six modules of the FINESS system

Chapter 11

Results

There may be programs which can beat anyone at chess, but they will not be exclusively chess players. They will be programs of *general* intelligence, and they will be just as temperamental as people. ‘Do you want to play chess?’ ‘No, I am bored with chess. Let’s talk about poetry.’ That may be the kind of dialogue you could have with a program that could beat everyone.

— DOUGLAS HOFSTADTER
Gödel, Escher, Bach

Victory goes to the player who makes the next-to-last mistake.

— CHESSMASTER SAVIELLY GRIGORIEVITCH TARTAKOWER

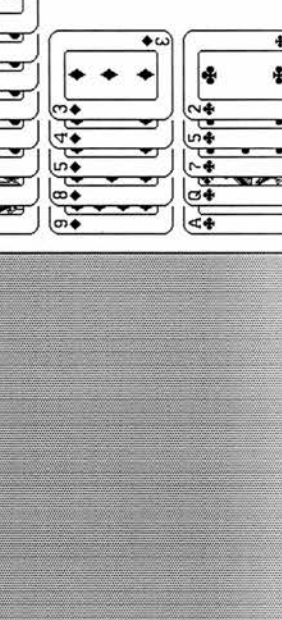
In this chapter, we attempt to give a flavour of the kind of results that FINESSE is capable of producing. We present and describe examples of plans formed for single-suit problems (§11.1) and for global problems (§11.2). A more extensive suite of test results using problems drawn from the Bridge Encyclopedia [ACBL 94] is given in Appendix C.

All the plans presented in this chapter are simply produced using FINESSE’s top-level interface, shown in Figure 11.1 on the following page.

11.1 Single-suit Plans

We begin by describing FINESSE’s solutions to a number of sample card combinations. These illustrate the way in which the best lines of play are selected by the system and also demonstrate the reduction in search produced by the restriction to tactics. We

Dealer:
West



Tricks Taken:
North/South:
East/West:

Hand 1 (North):

- ♠ 5, 4, 3
- ♥ 8
- ♦ 7, 6, 2
- ♣ 9, 8, 7, 6, 5, 4, 3, 2

Hand 2 (South):

- ♠ 9, 8, 7, 6, 5, 4, 3, 2
- ♥ 9, 8, 7, 6, 5, 4, 3, 2
- ♦ 9, 8, 7, 6, 5, 4, 3, 2
- ♣ 9, 8, 7, 6, 5, 4, 3, 2

Hand 3 (East):

- ♠ 9, 8, 7, 6, 5, 4, 3, 2
- ♥ 9, 8, 7, 6, 5, 4, 3, 2
- ♦ 9, 8, 7, 6, 5, 4, 3, 2
- ♣ 9, 8, 7, 6, 5, 4, 3, 2

Hand 4 (West):

- ♠ 9, 8, 7, 6, 5, 4, 3, 2
- ♥ 9, 8, 7, 6, 5, 4, 3, 2
- ♦ 9, 8, 7, 6, 5, 4, 3, 2
- ♣ 9, 8, 7, 6, 5, 4, 3, 2

Game Options

- ☐ Spades
- ☐ Hearts
- ☒ Diamonds
- ☐ Clubs

Single Suit Options

- Plan
- Trace Plan
- History Plan
- Interpret
- Top Tricks

Game Options

- Help
- Play Game
- Quit All

Welcome to XFinesse, Version 1.0.0
OK, game state loaded from /home/dream3/ianf/phpProlog/NewGames/a1
Planning a suit with history depth 5... OK. Interpreting... OK.
This leads to 5 tricks under any distribution of the outstanding cards.

Saved Games

- dl1
- ashortdemo
- debug
- demo1
- demo2
- demo3
- easy
- enc39
- except
- fin1

also give some examples of the textual explanations FINESSE produces to describe the chances of success of the optimal lines of play.

11.1.1 Basic Performance

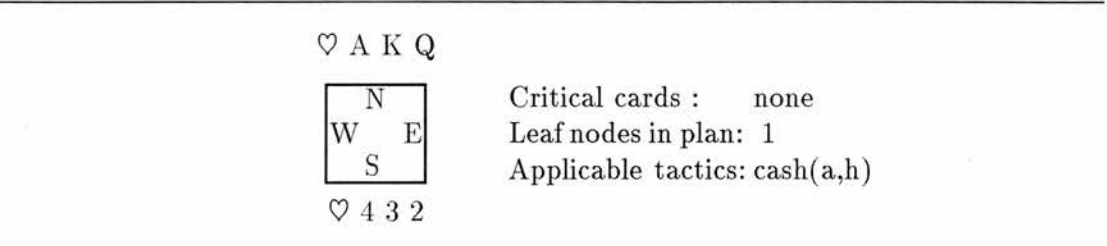


Figure 11.2: Simple card combination with no losers

In the simple example of Figure 11.2, declarer has three masters, so he is sure of winning three tricks. Since his longest hand only contains three cards, it should also be obvious that there are also no losers in the suit. This means that the only applicable tactic is the cash tactic, and that the defenders’ plays are immaterial (assuming no trump suit). The tree produced for this plan is therefore completely linear, whereas the actual number of legal play sequences¹ is 1,451,520.

The interpreter’s analysis of the payoff offered by the cash tactic in this situation is that ‘This leads to 3 tricks under any distribution of the outstanding cards’. Note that if an extra low card is added to one of the declarer’s hands, however, FINESSE ‘discovers’ for itself that it can gain an extra trick when the outstanding cards are split 3–3. The plan generated for such a situation is shown in Figure 11.3 on the next page (note that in this figure, Aces are represented as the number 14, Kings as the number 13, and so on). The path on the far right-hand side of this tree corresponds to a play sequence in which the defenders play low cards each time the declarer cashes a master, so that eventually the declarer’s last card (in this case, the 8) can win a trick. The interpreter algorithm describes this line of play with the explanation ‘This leads to 4 tricks if the remaining six low cards are split evenly between East and West’, and gives it the correct probability of 0.355.

¹ The general formula for the minimum size of the search space when declarer has n_1 cards in one hand and n_2 cards in the other, with the defence holding d cards in the suit is $n_1!n_2!(d+1)!$.

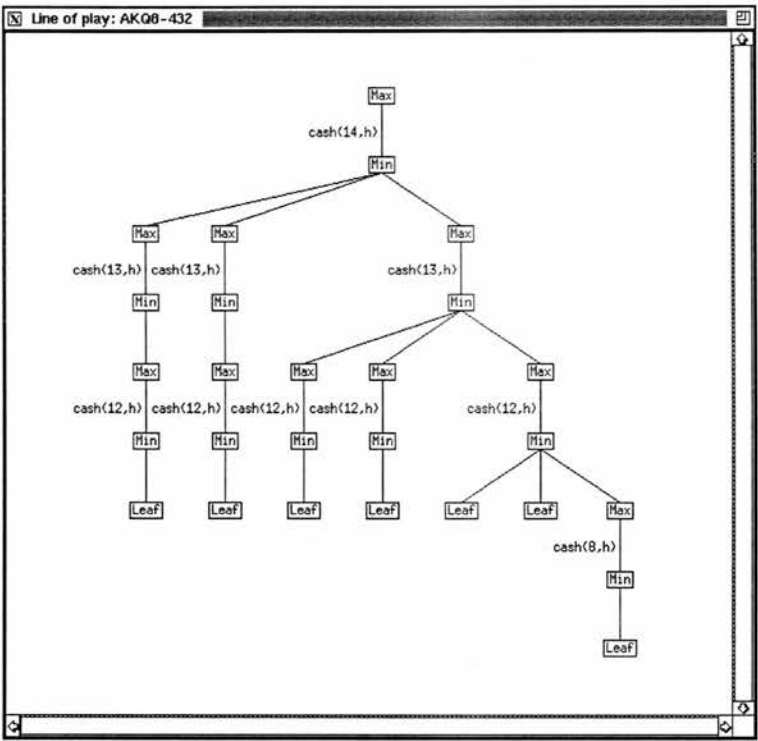


Figure 11.3: Plan which discovers chance of winning extra trick

The example combination of Figure 11.4 is a variation of the finessing theme we discussed in detail in Chapter 5 with the situation of the $AQ\spadesuit$ opposite the $2\spadesuit$. This time, however, the declarer has additional low cards and the Queen is also opposite the Ace. Whereas before FINESSE would select the finesse tactic, this time it selects the cash, as this produces two tricks not only when East has the King (a finesse is executed on the succeeding trick), but also when West has the singleton King. The

♠ A 6 4

N

W E

S

♠ Q 7 5

Critical card : King

Leaf nodes in plan: 75

Applicable tactics: cash(a,s)

finesse(3,east,q,s)

duck(s)

Figure 11.4: Card combination allowing finessing

interpreter algorithm evaluates the probability of success for this line of play to be 0.505, and gives the following explanation:

This leads to three tricks if East holds the King, or if West holds the singleton King.

FINESSE also copes well with positions in which a choice of finesses is available. For example, in the situation of Figure 11.5, FINESSE ‘discovers’ by itself the ‘principle’ that when it is possible to finesse more than one card in the opposite hand, the best chance of winning most tricks is by finessing the lower card first.

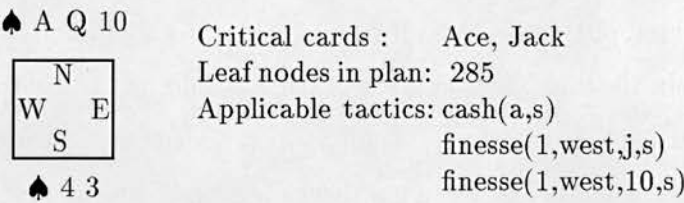


Figure 11.5: Card combination offering a choice between finesses

FINESSE correctly chooses to finesse the Jack, evaluating the probability of success to be exactly .24, giving the explanation:

This leads to 3 tricks if West holds the King and West holds the Jack.

Note that FINESSE also produces correct analyses in situations where declarer holds so many cards in a suit that the ‘principle’ of finessing the lowest honour does not apply.

11.1.2 Comparison Against Bridge Encyclopedia

Although there are many books on Bridge card play, relatively few authors commit themselves to specifying actual probabilities for the lines of play they discuss. However, one source which does give such probabilities is the Bridge Encyclopedia [ACBL 94]. As we mentioned in Chapter 6, this is also an example of a text which (implicitly) analyses lines of play under the assumptions of the best defence model of a game. It therefore supplies a good source of examples for comparison with the output of FINESSE’s single-suit planner.

In Appendix C we give a detailed summary of FINESSE’s performance on the problems from the Bridge Encyclopedia. Note that the default operation of FINESSE’s interpreter algorithm is to find lines of play with the highest chance of making the *maximum* number of tricks in a suit (since this is the effect of the relationship we defined over profiles in Chapter 8). This differs slightly from the Bridge Encyclopedia which, whilst usually specifying such lines of play, sometimes also includes extra variations which are optimal if the goal is to make some number of tricks which is *less* than the maximum. In our comparison, we concentrated on FINESSE’s basic performance, considering just the lines of play for achieving the maximum number of tricks. The main result of this comparison was that if the entire problem set was to be played with randomly distributed outstanding cards, the expected number of cases where FINESSE would fail to obtain the maximum number of tricks would be 11 (approximately 1.7%). However, a surprising amount of non-locality was revealed, with FINESSE selecting a sub-optimal line of play in 161 cases — a figure which includes the 57 problems which were corrected by the non-locality heuristic described in Chapter 7. On average, sub-optimal lines of play miss the maximum number of tricks with a probability of 0.07.

A good illustrative example of FINESSE performing correctly is the situation of Figure 11.6. The best line of play for making 5 tricks with this card combination is



Figure 11.6: A card combination from the Bridge Encyclopedia

described in the Encyclopedia as follows:

Play off the top honours (36%). This is fractionally better than the immediate finesse.

FINESSE’s analysis agrees with this assessment, giving a probability of 0.363 to the chance of making five tricks by playing the honours, and 0.359 to the probability of success with the finesse.

A further example in which the aim is to make five tricks is shown in Figure 11.7. For

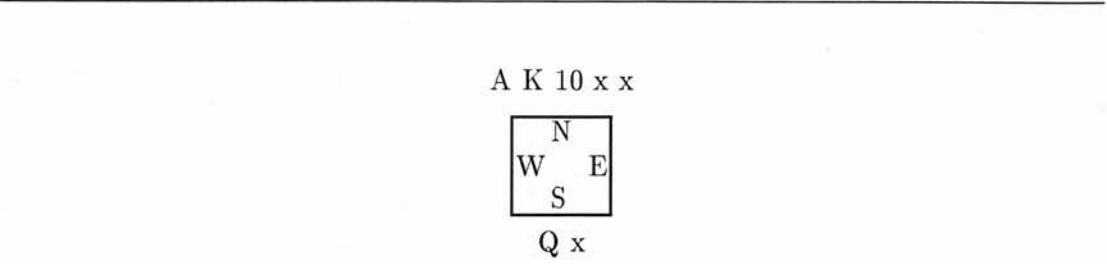


Figure 11.7: A second card combination from the Bridge Encyclopedia

this combination, the Encyclopedia gives the best line of play as being ‘play off the top honours (52%)’. FINESSE agrees with this percentage chance of success but in fact distinguishes between the playing of the Ace and the playing of the Queen. To see why this is so, consider the profiles of the two lines of play:

cash the Ace	[3▷0.136, 4▷0.347, 5▷0.517]
cash the Queen	[3▷0.129, 4▷0.355, 5▷0.517]

Whilst both have equal chances of making five tricks, the line of play beginning with the Queen actually has a higher chance of making *four* tricks. Thus FINESSE prefers to start with the Queen rather than the Ace in this situation.

11.1.3 Discovery of Errors

FINESSE also discovered an error in the Encyclopedia, arising from the situation depicted in Figure 11.8 (Problem 26 in the Encyclopedia).

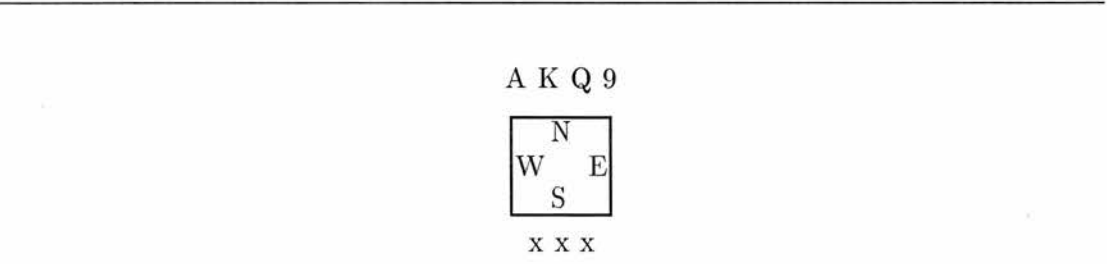


Figure 11.8: Card combination number 26 from the Bridge Encyclopedia

The obvious line of cashing the top masters in this situation produces 4 tricks with a

probability of 0.419, as shown in Figure 11.9. There is a modification to this basic line

Split	Probability of split	Distributions	Cases	Total Prob
6-0	.007	JTxxx-v	1/1	.007
5-1	.076	Jxxxx-T Txxxx-J	2/6	.025
4-2	.242	xxxx-JT	1/15	.016
3-3	.355	all	20/20	.355
2-4	.242	JT-xxxx	1/15	.016
1-5	.076	—	—	0
0-6	.007	—	—	0
			Total	0.419

Figure 11.9: Distributions allowing four tricks to be taken, and their probabilities

of play, though, in which the nine is finessed if East plays an honour on the Ace or King. This new line of play is superior to the original whenever East starts with Tx or Jx. However, it wins one less trick in the cases where East holds JT_x. Thus, the probability of winning four tricks with this line of play differs from the probability of winning four tricks when simply cashing masters by:

$$\begin{aligned} & prob(Jxxx-Tx) + prob(Txxx-Jx) - prob(xxx-JTx) = \\ & 4/15 \times .242 + 4/15 \times .242 - 4/20 \times .355 = 0.058 . \end{aligned}$$

This second line of play therefore has a probability of 0.48 of making four tricks, and is correctly identified as the optimal choice by both FINESSE and the Encyclopedia. The Encyclopedia, however, also gives the similar problem of Figure 11.10 (Problem number 31 in the Encyclopedia) which it claims has the same analysis. In this example,



Figure 11.10: A card combination analysed incorrectly in the Bridge Encyclopedia

however, it is no longer possible to take 4 tricks in the case where West has all the

outstanding cards. The chances of making four tricks in this situation is therefore 0.47, and not 0.48, as listed in the Encyclopedia.

FINESSE also discovered the following error (first reported in [Frank 91]) in another expert text, arising from the situations of Figure 11.11.

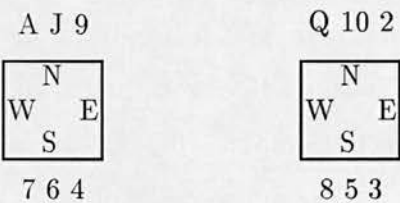


Figure 11.11: Card combinations analysed incorrectly in an expert text

In [Reese & Dormer 91, Page 20] these situations are analysed as follows:

As usual, declarer begins by finessing the lowest card of the combination. In the first example he finesses the 9, and if this brings forth the King or Queen there is a good chance that a finesse of the jack will win on the next round. In the second example, the chances of making just one trick by leading up to the 10 first are similar. In each case success depends on the combination of an even chance with a three to one on chance — $37\frac{1}{2}$ per cent.

When FINESSE is given these situations, however, the selected line of play is the same, but the probability of success is different — 0.377. The original version of FINESSE found this discrepancy because the ordering of the methods in its database produced this line of play first, and the probability of success subsequently calculated for it was found to be different to that given by Reese. The current version, however, actually searches the entire meta-level space and then selects this line of play as the best option. Furthermore, the current version also produces the following explanation for its choice:

This leads to 2 tricks if West holds at least one of the King and Queen and West holds the ten, or if West holds the doubleton King, Queen.

This explanation makes it an easy matter to check that it is Reese who has produced the incorrect analysis, having overlooked the final case of the doubleton King, Queen.

11.1.4 Relaxing the Best Defence Model

One interesting observation to emerge from the comparison with the Bridge Encyclopedia's solutions was that occasionally the Encyclopedia would discuss the possibility of using the cards played by the defenders to make inferences about the actual state of the world. For example, in the situation of Figure 11.12, the best line of play under the assumptions of best defence is found by FINESSE and by the Bridge Encyclopedia to be playing off the masters. However, the Encyclopedia adds that 'against defenders who would not false-card from the J10x, cash the Ace and finesse the nine if an honour appears from East'. This modified line of play has an approximately 2% better line of success than simply cashing the winners.

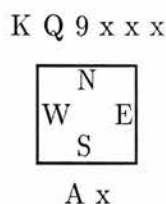


Figure 11.12: Combination where relaxing A-II can improve chance of success

In the true best defence model of a game, MIN chooses a strategy after MAX (assumption A-II). Thus, such inferences will never gain an advantage for the declarer: if he builds a strategy based on the expectation of the defenders not false-carding, they will false-card. For instance, in the above example declarer would lose a trick if he chose a finesse based on the incorrect assumption that a Jack played from the J10x by East was not a false-card. However, against defenders who *are* predictable in some situations it can be possible to gain an advantage by learning their tendencies. The fact that sometimes human players make use of such inferences (as demonstrated by the examples from the Bridge Encyclopedia) indicates that their incorporation may be a useful addition to a computer Bridge system.

Note that the Encyclopedia also occasionally includes solutions which involve a relaxation of assumption A-I that the defenders have perfect information. For example, it describes the solution to the situation of Figure 11.13 (Problem number 330 in the

Encyclopedia) as ‘Lead small from the ace-queen, in case East has the singleton king; then finesse the queen [50%] (and if East panics into playing the King from Kx [54%]).’

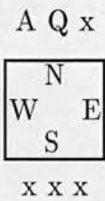


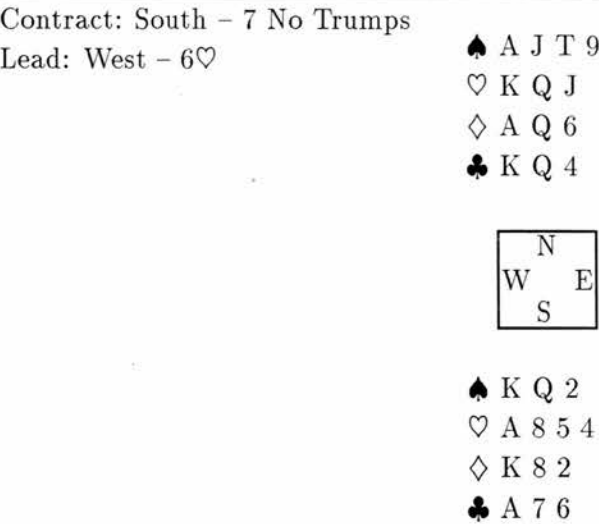
Figure 11.13: Combination where relaxing A-I can improve chance of success

We have already discussed a situation similar to this in Chapter 5 (Figure 5.9), where we saw that FINESSE uses the duck tactic to represent the best play. FINESSE, though, only assesses the probability of success for this line of play as 0.505, since East would clearly not make the mistake of playing the King from Kx under the assumptions of best defence. Incorporating the ability to reason about possible mistakes by the defenders, however, would be a useful step towards selecting between lines of play on the basis of which is *hardest to defend against*, as we discuss in §12.2.11.

11.2 Global Plans

FINESSE has successfully produced solutions to numbers of global problems. In Chapter 9 we have already discussed a successful interleaving of a squeeze situation. Here we give some further examples.

We begin with the very simple situation of Figure 11.14 on the next page. Faced with this problem, FINESSE’s planning loop begins by finding the top tricks lines of play and the best lines of play for each suit. In fact, each suit’s top tricks and best lines of play are the same for this deal, so there will only be four lines of play that are distinct. The only suit in which there is any resource requirement is the heart suit, so FINESSE selects a supplier (it picks the club suit) and constrains it to be interleaved with the heart suit. This constraint enables the solution constructor to produce a solution which guarantees 13 tricks. Figure 11.15 on Page 274 reproduces the top portion of the interleaving produced by the solution constructor for this global plan. This shows how



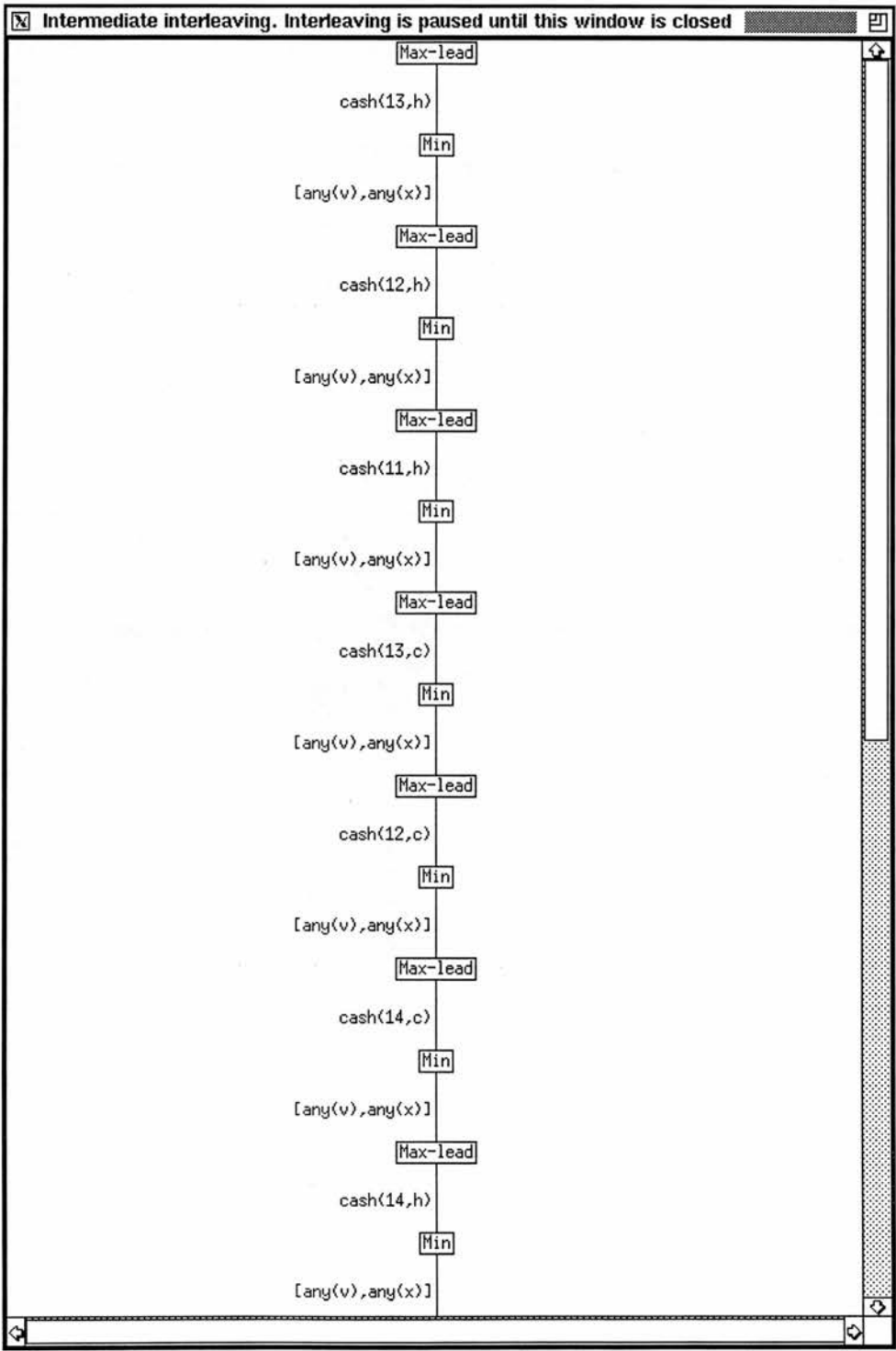


Figure 11.15: Top portion of a solution constructor interleaving for Figure 11.14

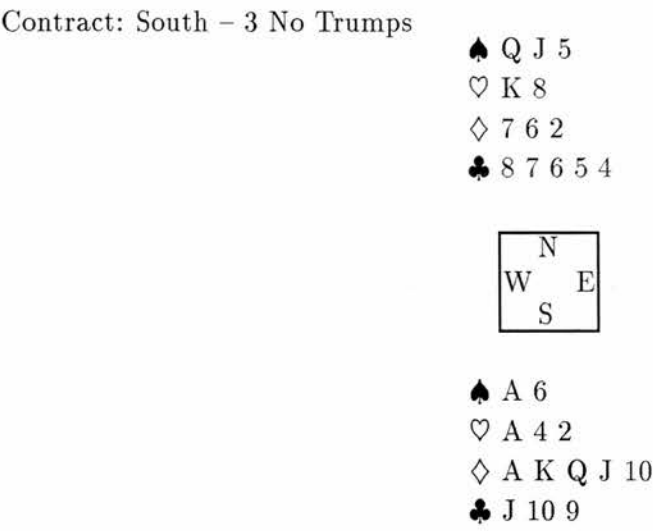


Figure 11.16: An example Bridge deal, repeated

consumption, producing a ‘worst best’ global plan with a profile of $[8 \triangleright 0.996, 10 \triangleright 0.004]$ (there are eight masters in total and if the $K\spadesuit$ is singleton, two extra tricks will be won by cashing the $Q\spadesuit$ and the $J\spadesuit$).

FINESSE next tries to improve on this plan by interleaving the best line of play for the spade suit (as determined by the interpreter algorithm) with the top tricks lines of play for the remaining suits. As we saw in Chapter 8, this improved plan for the spade suit guarantees an extra trick but requires an entry into the North hand. FINESSE correctly constrains the heart suit to supply this resource, enabling the solution constructor to construct the line of play partially depicted in Figure 11.17. From this figure, it can be seen that initially the $A\spadesuit$ is cashed, followed by the $Q\spadesuit$ and $J\spadesuit$, if the King drops under the Ace. If the King doesn’t drop, a low card is led towards the $Q\spadesuit$ (the `sequence(12,s)` tactic) followed by the cashing of the $K\heartsuit$ to re-enter the North hand. The profile of this interleaving is $[9 \triangleright 0.996, 10 \triangleright 0.004]$, and since this is \geq than the profile already produced, this interleaving is retained as the sole best solution so far. (Note that FINESSE does not currently take account of an opening lead in a contract, but the presence of the interleaving constraints on the $K\heartsuit$ would cause the opening trick to be won with the $A\heartsuit$, and not the $K\heartsuit$, if hearts was the suit chosen by the defence at trick one.)

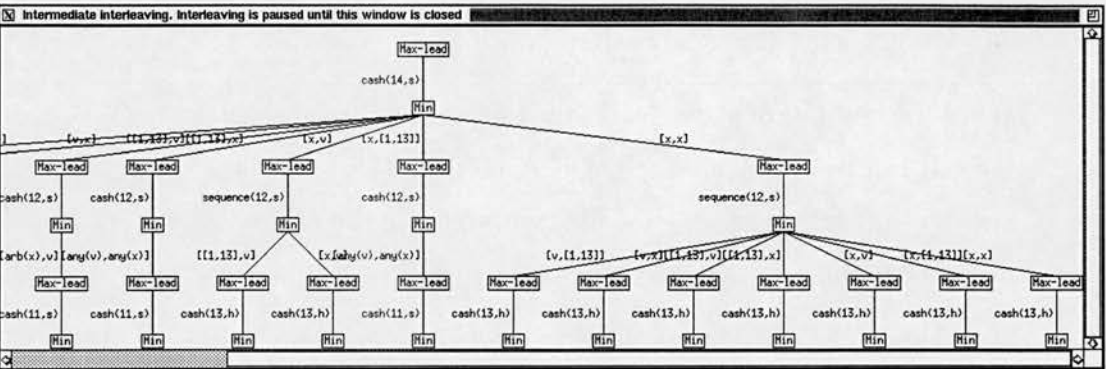


Figure 11.17: Top portion of a solution constructor interleaving for Figure 11.16

FINESSE now backtracks to try other combinations of lines of play. The plan sets where just the heart or diamond suits are improved are not interleaved because their best-case profiles are found to worse than the interleaved profile already generated. This is not the case when it comes to improving the club suit, however, in which two extra tricks can be won by playing cards until the Ace, King and Queen have been cashed by the defence. Indeed, the best-case profile for the set of plans containing this improved line of play for the club suit is that ten tricks will be won with a probability of 1. However, when the solution constructor attempts to interleave a global line of play from this plan set, it discovers that the clubs cannot be developed without giving the defenders the opportunity to take extra tricks by attacking the heart suit. The overall plan which is finally selected, then, is the one which uses the spade suit to develop an extra trick.

The way in which a simple plan is initially produced and then gradually improved upon is further illustrated by the example of Figure 11.18 (first presented in Figure 8.4 on Page 194). As we saw in Chapter 8, the best line of play in this situation is to use the finesse of the $Q\heartsuit$ and the subsequent cashing of the $A\heartsuit$ to supply the two entries that are needed to establish the spade suit. This plan produces 12 tricks if the $K\heartsuit$ is with West, and six tricks otherwise. Before generating this optimal plan, however, FINESSE starts with the top tricks line of play for each suit. The profile produced by the solution constructor when interleaving these lines of play is $[6 \triangleright 0.996, 7 \triangleright 0.004]$. This is because there are six top masters which will always take a trick (1 in the heart suit, 3 in diamonds, and 2 in clubs), and an extra trick will be developed whenever the

$K\heartsuit$ is singleton. The almost solid spade suit actually makes no contribution to this interleaving, since it has no masters which can be cashed.

When the best line of play replaces the top tricks line of play for the spade suit, however, the suit can be attacked using sequence tactics. This is the plan set which is examined next by FINESSE, producing the profile $[6 \triangleright 0.996, 12 \triangleright 0.004]$, since if the $K\heartsuit$ is singleton two entries will be available into the North hand to supply the resource requirement of the improved line of play for the spade suit. A screen capture of the initial levels of the solution constructor tree for this interleaving is given in Figure 11.19, showing how the $A\heartsuit$ is first cashed and then the $K\spadesuit$ is played.

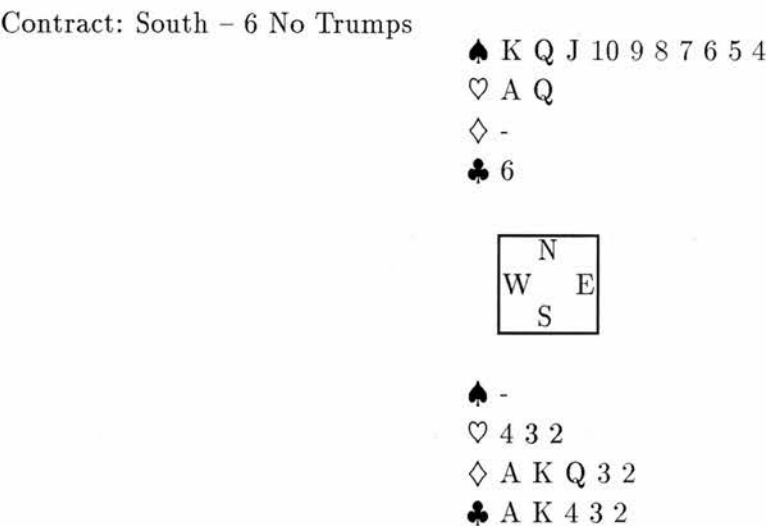


Figure 11.18: A possible deal containing an almost solid suit

The next combination of plans examined by FINESSE consists of the top tricks line of play in all the suits except the heart suit. Since the best single-suit solution in hearts is to finesse the Queen, this interleaving has the profile $[6 \triangleright 0.5, 7 \triangleright 0.5]$. FINESSE then continues to examine other combinations of top tricks and best lines of play but finds their best-case profiles are inferior to the profiles of the plan objects generated already until the optimal combination (consisting of the improved lines of play for the spades and hearts suits together with the top tricks lines of play for diamonds and clubs) is encountered.

To give an impression of the size of the trees that the solution constructor can produce

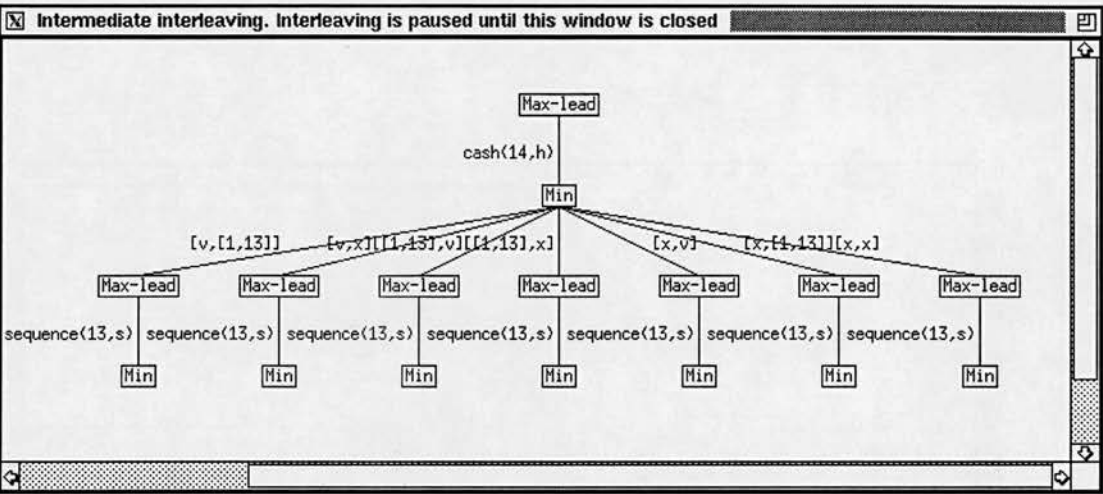


Figure 11.19: Top portion of a solution constructor interleaving for Figure 11.18

we also reproduce in Figure 11.20 a portion of the interleaving of the squeeze tactic discussed in Chapter 9. This screen capture shows a number of the leaf nodes that can be reached under a global plan which first tests the heart suit and then (if necessary) attempts a squeeze. Notice that three nodes can be seen where a squeeze tactic is applicable. In the left-most cases, an extra trick cannot be won if the squeeze fails. In the right-most case, however, some of the possible worlds under which the node can be reached will produce an extra trick even when the squeeze conditions are not satisfied: that is, the spade suit may still break 3-3. The three branches visible on the far right represent plays of the game in which the squeeze is not necessary because the heart suit was found to break 3-3 (there are also branches where a squeeze cannot be played because opposite defenders show out in spades or hearts, but none of these branches are visible in the figure).

Finally, note that FINESSE’s node preference reasoning allows it to solve situations such as that of Figure 11.21. Here, needing to make nine tricks, declarer has eight top tricks and two possible sources for the ninth. If the missing diamonds are split 3-3, then the first three rounds of diamonds will force them all to be played and the fourth diamond in North’s hand will provide an extra trick. Alternatively, a trick could be won with the Queen of spades if the King is with East.

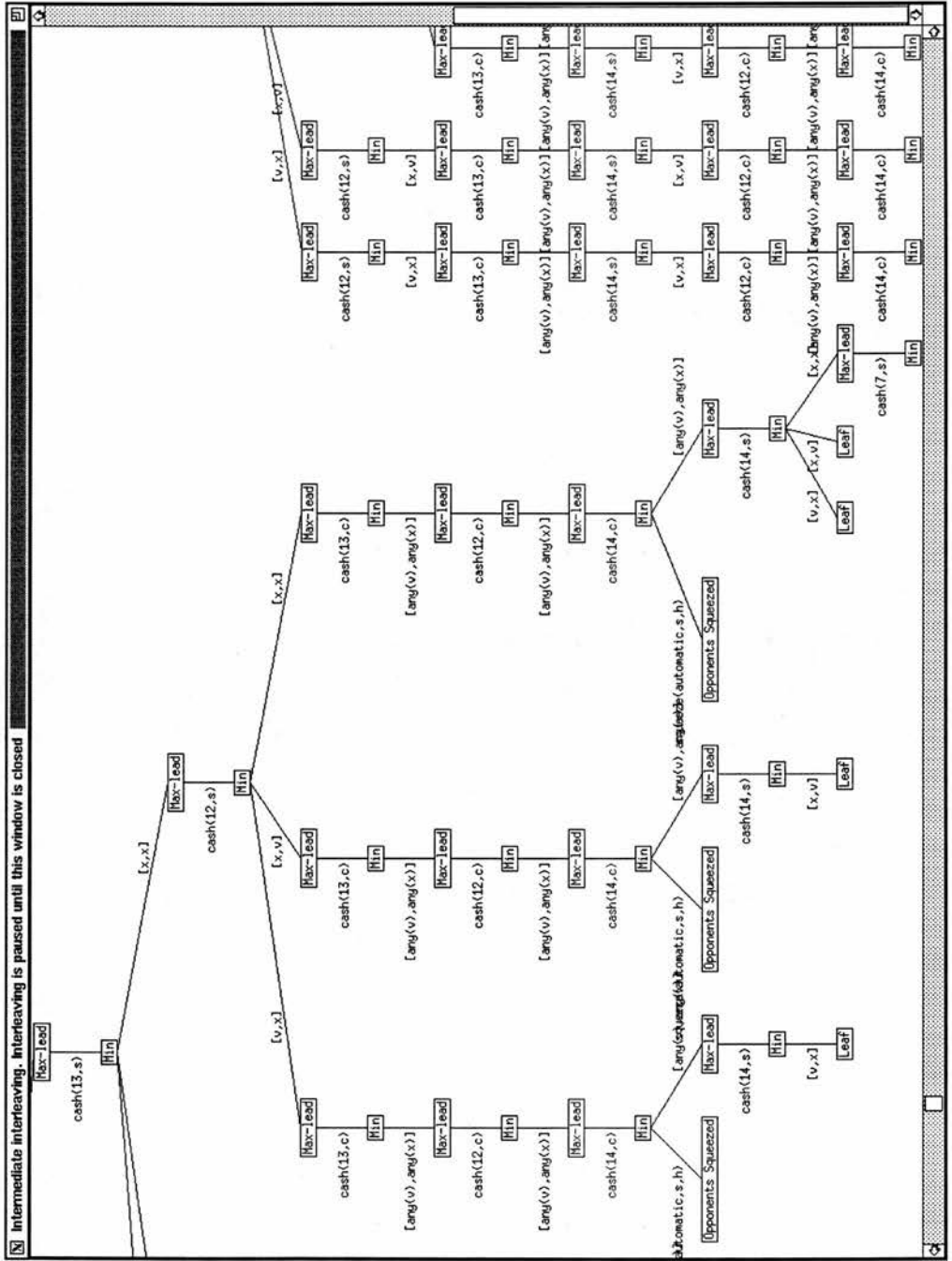


Figure 11.20: Portion of solution constructor interleaving including a squeeze

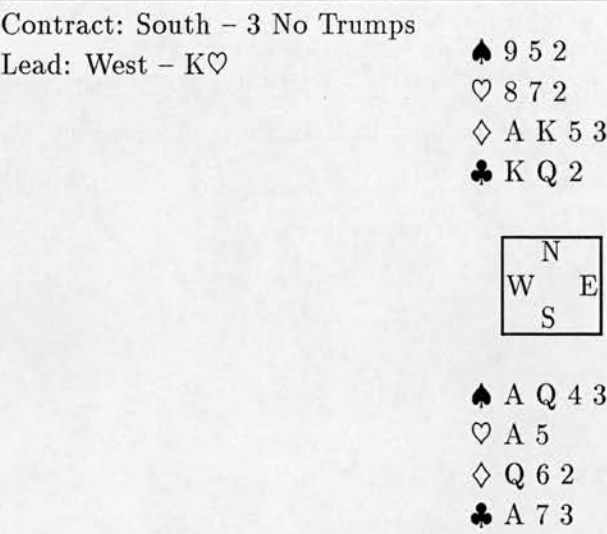


Figure 11.21: An example deal from [Berlin 85]

In order to form a plan for this hand, it is necessary to realise that it is possible to play the diamond suit before the spades, but not the other way around. For if the spade finesse were tried first, and were to lose, there would be a danger of the defenders taking four or more heart tricks and breaking the contract.

[Berlin 85] proposes a general agenda-based planning system which he claims could deal with this kind of situation. The two plans are placed on an agenda together with a task to select between them. The procedure associated with this task first attempts to order the alternatives so that if one branch fails, the other can still be tried. In this particular case, it is easily found that if the diamonds are tried first and don't split, the spade finesse can still be attempted. FINESSE, on the other hand, realises that control must not be surrendered to the defence and therefore increases the preference of lines of play whose resource analyses do not involve giving the lead away to the defence. The number of leads given to the defence can be found by examining the resource profiles of the lines of play, and thus the diamond suit is selected ahead of the spade finesse.

11.3 Summary

FINESSE has proved to be a capable Bridge system. In single-suit situations it consistently identifies a best line of play that matches or is close to the recommended lines of

play found in the Bridge literature. Also, the system is fully capable of finding plans that are optimal in complete deals. Moreover, FINESSE can justify the selection of any line of play with a probability profile describing its chances of success, and also with a textual explanation — abilities which have led us to find errors in the analyses of human experts.

Chapter 12

Conclusions

I believe cards will continue to thrive because they relate so closely to the human condition... so long as anyone is left to wield a card there will always be intelligent games reflecting the basis set-up of life, which starts us all off from unplanned and unequal opening positions, and itself is nothing if not the ultimate game of 'imperfect information'.

— DAVID PARLETT
A History of Card Games

True luck consists not in holding the best of the cards at the table:
Luckiest he who knows just when to rise and leave.

— JOHN MILTON HAY
Distichs, no. 15

We hope we have shown, in the foregoing pages, that there are real lessons to be learned from the study of games with incomplete information, and in particular from the study of Bridge. Now, as our time to rise and leave approaches, we conclude by summarising our main contributions, and by signposting some possible directions for further research, which we hope may both help and encourage those after us who are tempted to try their luck at the table.

12.1 Contributions

Mirroring the breakdown given in Chapter 1, we can categorise our contributions into the four general areas of search, planning, proof-planning, and Bridge.

12.1.1 Search

We considered in detail the problem of search algorithms in domains with incomplete information and an opposition. By formalising a best defence model of two-player games we showed that adaptations of minimax-like algorithms could not be guaranteed to produce optimal strategies in such domains. In particular, we examined double-dummy Bridge architectures and found that they suffered from the problems of strategy fusion and non-locality.

Non-locality, being a problem that occurs when competing agents in a domain have different beliefs about the state of the world, affects any algorithm which attempts to make choices without reasoning about the possible worlds under which a particular position may be reached. We identified an exhaustive strategy minimisation algorithm capable of identifying optimal strategies in two-player games, but saw that its complexity was doubly exponential in the number of a player's moves in the game. We were able to identify some general algorithms with lower complexity that dealt with some of the effects of non-locality, but their results were not guaranteed to be optimal.

In the game of Bridge, we showed that the best defence model of a game corresponded to the form typically analysed in expert texts, and saw that non-locality was a real problem that occurred in actual Bridge situations. We also showed how it was possible to tackle the effects of non-locality with domain-specific heuristics which, thanks to the high-level representation of the possible plays afforded by tactics, allowed possible worlds to be ruled out in certain situations.

12.1.2 Planning

We took Kambhampati's recent formalisation of plan-space planning as refinement search and used it as a framework to describe an architecture for interleaving lines of play. The implementation of this system provides support for the cliché-based approach of identifying formal structures describing the nature of a domain, and in particular demonstrates the feasibility of using resource-based reasoning to constrain the ordering of actions with disjunctive effects.

Also, we showed how the tractability refinement of using a total ordering could be

instrumental in tackling the problem of non-locality. Interleaving involves choosing an *order* for some given operators rather than actually choosing the operators themselves. Thus, by assuming that the auxiliary constraints specified all the possible interactions, we were able to simply consider the chance of success of a *particular* ordering obeying the auxiliary constraints. Non-locality is avoided since no operators are actually chosen within the solution constructor function, although the cost of constructing the solution constructor's tree is still exponential in the length of the game.

We pointed out that the above argument also provides a justification for the approach of splitting the global task of planning Bridge play into the subproblems of forming lines of play in individual suits. Given that exhaustive strategy minimisation is the only optimal solution we know of for incomplete information games under best defence, both the global problem of Bridge card play and the subproblems of single-suit play involve complexities that are doubly exponential in the number of a player's moves. It therefore makes sense to reduce the exponent by considering the shorter-length play sequences involved in single suits, and then just incur a cost which is exponential in the length of the game when interleaving.

By casting our planning algorithm within the classical plan-space planning paradigm we have shown how our techniques differ from, yet may be incorporated into, existing planning architectures. Domains in which we would expect our ideas to be most useful would be those with a sufficient degree of incomplete information to make examination of all the possible states impractical, yet in which the uncertainty could be quantified and expressed in some formal representation language.

12.1.3 Proof-planning

From a proof-planning perspective, the FINESSE system has provided us with some valuable insights. To date, in addition to finding proofs for mathematical theorems, proof-planning ideas have been successfully applied to computer system configuration [Lowe 91], and to program verification [Bundy *et al* 90a]. In implementing FINESSE, we have tackled many new problems including the need to cope with an opposition, and the need to represent and reason about domain actions with disjunctive effects. Besides the general contribution of demonstrating the applicability of proof-planning ideas in

a hard domain, we also drew lessons from the modifications which were necessary. In particular, we found that we were forced to change the representation of methods to allow the effects of operators to be inferred by domain rules.

The introduction of inter-suit tactics also showed the feasibility of a two-stage application of the proof-planning paradigm: one stage in which subproblems are tackled by specifying the possible operators as tactics, and a further stage where operators neglected by the original problem breakdown are re-introduced.

12.1.4 Bridge

In terms of the game of Bridge itself, we produced new lower bounds on the size of the search space involved in declarer play. These lower bounds were tighter than those produced by previous authors, providing, together with the formalisation of the phenomenon of non-locality, an explanation for the poor performance of computer Bridge programs to date.

Also, the act of specifying FINESSE's methods brought us to a better understanding of the game of Bridge itself. We saw in Chapter 5, for instance, that the Bridge literature typically 'explains' standard concepts such as finessing and ducking largely by the use of examples. The preconditions of the methods in FINESSE make explicit the 'knowledge' contained in numerous examples from the Bridge literature.

FINESSE proved to be a capable system, identifying global lines of play for No Trumps situations that concurred with those given in expert texts, and in the single-suit case even revealing errors in the analyses of some authors. The implementation of the system also addressed some of the problems that other authors have identified with automating Bridge card play. For example, [Stanier 75, Page 375] lists a number of reasons why a tree-searching approach is less suited to Bridge than to other games such as chess. Perhaps a good way to summarise some of FINESSE's features is to briefly outline how it addresses each of Stanier's objections:

- Stanier argues that any Bridge-playing program should not ignore the body of knowledge gained by Bridge-players in the past decades: FINESSE incorporates this knowledge in the declaratively encoded preconditions of its methods.

- Stanier is concerned that the probabilistic results returned by an evaluation function would make minimaxing a complex task: by building up a qualitative picture of the distributions that would lead to the success of each line of play in a suit, FINESSE avoids the loss of information inherent in a probabilistic approach.
- The use of the proof-planning paradigm controls search in a way that avoids unpleasantness such as the ‘horizon effect’ (there is no horizon, since the whole tree is searched).
- Stanier argues that a successful Bridge-playing program will need to have concepts such as ‘finesse the Queen of clubs’: these are exactly what is represented by the tactics in FINESSE.

12.2 Further Work

As it stands, FINESSE is clearly a long way from threatening to relieve Zia Mahmood of six-figure sums of money. We therefore conclude by highlighting some of the unresolved research issues which have been raised by our work, in approximate increasing order of difficulty.

12.2.1 Play Module

Since the output of FINESSE corresponds to a complete line of play with a profile describing its chances of success, a play module was not *necessary* in order to judge the system’s performance. However, in order to compare it directly with other Bridge programs, which do not have this ability to generate complete plans at the start of a game, a play module will be required.

12.2.2 Incorporating Information From the Bidding

The qualitative nature of the uncertainty representation used in FINESSE makes it fully capable of utilising information inferred from the bidding stage of the game, or from the opening lead. For instance, any possible worlds which are found to be incompatible with the bidding history of a game could be removed, using subsumption, from all the \mathcal{C} -conjunctions generated by the interpreter or interleaver algorithms. A basic

bidding system has been incorporated into the FINESSE architecture [Asher 93] and the ability to make inferences about any player's bids also added [Green 95]. However, the language in which these inferences are made is different from the \mathcal{C} -conjunctions used by FINESSE, so to make use of them a translation algorithm will be necessary.

12.2.3 A Bridge Tutoring System

The high-level nature of FINESSE's plans, combined with the ability to produce textual explanations, makes the system a promising basis for an automated Bridge tutor. Such a system would allow a user to test and develop their card-play ability by comparing the user's line of play with the best line of play selected by FINESSE, and critiquing the user's moves either as they are made or at the end of a play problem.

The exact design of a tutoring system would depend on the consideration of a number of alternative interface issues. For example, for each card-play the user could either be presented with a list of sensible possibilities, or be allowed a free choice. The list of possible plays could also be in the form of tactics or in the form of individual card plays. Further, comments could be made about each of the user's moves, or alternatively only when they deviate from the optimal line of play. Other useful features would be to check the user's estimation of the chances of the success of their line of play against that produced by FINESSE, and to help beginners by describing their chosen card-plays in the higher level language of tactics. Explanations of some of the basic rules and concepts of Bridge such as following suit and the notion of finessing, might also be profitably incorporated.

The utility of a tutoring system would be further enhanced by improving FINESSE's ability to justify its choice of best plan. FINESSE can already produce textual explanations of why a particular line of play is selected, such as 'Finesse the Queen: this leads to two tricks when the King is with West,' but in a tutoring system, it would be useful to allow the user to query such explanations with the simple question 'Why?' (*e.g.*, by clicking on a button). The system should then use the plans generated by FINESSE to demonstrate why the chosen line of play is optimal. For example, the explanation above would be illustrated by generating one situation where the King is with West and one where the King is with East, and allowing the user to examine the subsequent

play with all four hands visible. The cards which are of no consequence to the outcome of the play should be replaced with 'x's to concentrate attention on the significant features. An important part of this extension would be the automatic generation of card distributions between the defenders to be used as *exemplars* when explaining the plan.

12.2.4 Extension to Suit Play

In the interleaving architecture discussed in Chapter 8, the possible effects of a trump suit were largely ignored. In particular, the possibility that a tactic could lose to a trump played by a defender was ignored. Including this possibility will increase the cost of running the solution constructor function, as the number of branches required to describe the possible plays by the defence in any state will increase. However, the principle of the solution embodied by the solution constructor function will remain the same.

12.2.5 Adding Inter-suit Methods and Tactics

In Chapter 9 we demonstrated the feasibility of squeeze tactics by giving an example tactic and interleaving. Extending the repertoire of inter-suit tactics to account for other possible plays involving more than one suit would be an important direction for further research. One obvious example of such a tactic missing from the No Trumps FINESSE system is ruffing. The incorporation of any inter-suit tactics into the planning architecture will require the specification of methods to identify the situations in which they will be applicable.

12.2.6 Improving the Top-level Control Structure

We noted in Chapter 10 that the control over backtracking in FINESSE's top-level planning loop was rather primitive. Although in many situations the best global line of play is composed of a combination of the 'best' lines of play and the 'top tricks' line of play from the individual suits, this is not always so. The planning loop should therefore be able to assess the trees of tactics in each suit using different relations to evaluate the possible payoffs when it requires to pursue goals such as developing extra

entries into a particular hand, or preventing a specific defender from obtaining the lead.

12.2.7 Making Inferences Based on Opponents' Play Styles

We saw in Chapter 11 that sometimes it can be possible to use knowledge of an opponent's style of play to make inferences about the actual state of the world. Also, if we assume that the opponents are intelligent, and have reasons for playing the way they do, we can try to hypothesise their reasons and thus gain more information about the actual state of the world. We pointed out that such inferences break the best defence assumptions that MIN chooses his strategy after MAX, but even so, the fact that sometimes human players make use of such inferences (as demonstrated by the examples from the Bridge Encyclopedia) indicates that their incorporation may be a useful addition to a computer Bridge system.

12.2.8 Identifying More General Algorithms to Deal with Non-locality

We pointed out in our discussion of non-locality that we know of no algorithm which identifies guaranteed optimal strategies more efficiently than exhaustive strategy minimisation. Establishing a lower bound on the complexity of this problem, or better still producing a more efficient algorithm, remain open problems for the design of programs which operate in domains like that of Bridge card play.

12.2.9 Planning to Discover Information

Our interleaving architecture produces plan objects that represent a *single* line of play for a complete game. Since we assume that the resource-based constraints impose a total ordering, these lines of play never contain choice points where two operators with different results can be selected between. This is how the problem of non-locality is avoided. However, some plans are difficult to generate in such an architecture. For example, we have encountered situations where, in order to increase the chance of making a correct decision (*e.g.*, between finessing or cashing), it is best to first play out winners in all the other suits to gain clues as to the position of the remaining cards.

FINESSE would currently have to represent such situations by creating two global lines of play, and monitoring plan execution to ensure that the one with the greater probability of success did not change. Whilst a more natural approach would be to allow a disjunctive goal to be passed to the interleaver, to do this would re-introduce the problem of non-locality. Some actual play situations do appear to be solved by humans by discovering information, however, and indeed FINESSE itself introduces some inferences about card positions which it *always* knows to be true when dealing with non-locality in single suits. Investigating how such choices based on gained information could be incorporated without re-introducing non-locality is an interesting research area.

12.2.10 Extension to Mixed Strategies

In discussing non-locality and giving our solution of exhaustive strategy minimisation we noted that the main limitation of our model was the restriction to pure strategies. In Bridge, this limitation was found not to be serious since many expert texts also give solutions to declarer play problems in the form of pure strategies. However, we pointed out that the most general solution would be in the form of a mixed strategy, to afford some protection against the opponents ‘finding out’ your intentions. Using the pure strategies generated by FINESSE as building blocks for generating mixed strategies would allow these more general solutions to be produced.

12.2.11 Choosing the Hardest Line of Play to Defend Against

Two lines of play with the same chances of success against best defence may actually be distinguishable: defending correctly against one may be significantly harder than defending against the other. This type of situation was encountered in the CHINOOK checkers-playing program (discussed in Chapter 2), which in one of its games against Marion Tinsley was able to use deep search to announce the game drawn on move 5! The contest, of course, proceeded in case the human would make a mistake and lose. However, [Schaeffer *et al* 93a, Page 56] explains that:

In this situation, the program should choose between drawing lines, trying to maximise the probability of the opponent making a mistake.

12.2.12 Extension to Defender Play

As a final extension, we should point out that a system capable of winning Zia Mahmood's million pounds would have to be able not just to play the cards as declarer, but also to take the role of a defender.

In a sense, there is a kind of 'chicken-and-egg' situation here: skillful declarer play requires an understanding of how defenders play, whereas good defence necessitates reasoning about what actions the declarer is likely to be taking. In forming plans for declarer play, we used the increased information resulting from having two hands visible and under our control to allow us to form plans under the assumption of best possible play in *each* possible world by the defenders. We saw that this type of analysis is also common in expert texts on Bridge. In defender play, however, two hands are still visible but only *one* can be controlled. This lack of control makes it difficult to reduce the number of possible actions using tactics. Instead, the goal of increasing the defending side's knowledge of each other's hands to the point where they can actually determine what tactics may be applicable becomes significant. We have seen, though, that once we start to reason about the beliefs of the other players, the cycle of reasoning becomes hard to stop.

Appendix A

An Overview of Commercial Computer Bridge Systems

I think there is a world market for maybe five computers.

— THOMAS WATSON
Chairman of IBM, 1943

Software developers, perhaps hoping to cash in on the market of 100 million Bridge players worldwide, have managed to produce a large number of commercial systems capable of playing the game. Here we list and describe some of these products, with the aim of enabling the better packages to be more easily identified. Figure A.1 summarises the various computer Bridge systems of which we are aware, showing the platforms on which they operate, whether they contain algorithms for defender play and declarer play, and whether a demonstration copy is available. Most of the systems cope with the entire course of the play of a hand, but we have also included some programs whose only concern is with the bidding phase.

The following sections present a brief overview of many of these programs, based on reviews published in Bridge or computer magazines, on our own personal experiences with the programs, or on a combination of both. These overviews concentrate wherever possible on the playing strength of the systems, although some information on the overall presentation and packaging is also often included. Clearly, there are too many systems for us to individually examine and play-test each one, so many of the overviews draw extensively on magazine reviews, especially those from Canadian Master Point [Lee 94]. In these cases, direct quotations are identified in the normal way, using either

Name	Platform	Defend	Declare	Demo?	Section
Amiga Bridge 1.06	Amiga	•	•		A.1
BBC Bridge Companion	Plugs into TV				
Bicycle Bridge	IBM	•	•		
Bidding	IBM			•	A.2
Bridge 7.0/Bridge 8.0	Mac/IBM				
Bridge Buff	IBM	•	•		A.3
Bridge Champion	IBM	•	•		A.4
Bridge For Windows	IBM	•	•		A.5
Bridge Master	IBM	•		•	A.6
Bridge Master	Mac	•	•		A.7
BridgeMate	IBM	•	•	•	A.8
Bridge Olympiad	IBM	•	•		A.9
Bridge Pal	IBM	•	•	•	A.10
Bridge Parlor	IBM	•	•		
Colossus Bridge	IBM/Amiga/Spectrum/C64	•	•		
Grand Slam Bridge I/II	IBM	•	•		A.11
Meadowlark Bridge	IBM(Windows)	•	•	•	A.12
Microbridge	IBM	•			A.13
Micro Bridge	IBM	•	•		A.14
Micro Bridge Challenger	IBM	•	•		
Micro Bridge Companion	Mac/IBM/Tandy	•	•	•	A.15
MVP	IBM(Windows)	•	•		
Omar Sharif's Bridge	IBM(Windows/Dos)/Mac/Amiga	•	•		A.16
Oxford Bridge 4	IBM	•	•		A.17
Positronic Bridge	IBM	•	•		A.18
Saitek Bridge Shadow	Hand-held	•	•		A.19
Saitek Pro Bridge 310	Hand-held	•	•		A.19
Saitek Pro Bridge 510	Lap-top	•	•		A.19
Radio Shack Bridge Companion	Hand-held	•	•		

in-line single quotes or an indented paragraph.

Wherever possible, details on availability and price have been included. (For readers in the UK, many of the programs can be obtained from Chess and Bridge Limited, 369 Euston Road, London NW1 3AR, Tel: 0171-388 2404, Fax: 0171-388 2407.)

A.1 BBC Bridge Companion

Price £199.99

This is a ‘tutorial’ system, and does not play random deals. During the bidding it can point out any mistakes and suggest alternatives. During the card play it lets you take the role of declarer, but stops you if you reach a position where the contract cannot be made, taking you back to the error. According to Forrester [Forrester & Dimbleby 95] it only allows ‘one or two lines of play’, and the display is rendered using ‘very basic—almost retro—Ceefax-style graphics’. Forrester’s summary is that the program provides a ‘decent basic tutorial’, but that it ‘is never going to explain even the basic subtleties of the game’, and that ‘you would get much more, much more cheaply, from a book’.

A.2 Bidding

Producer Platon M. Beletzky.
Address L. Pervomaiskogo st. 3, 13, Kiev, Ukraine, 252023.
Tel +7 (044) 2212794.
Email pmbel%gst.kiev.ua@relay.ussr.eu.net.
Price Shareware. Registration fee \$9.95.

This program allows the user to practise their bidding on randomly generated hands. During the bidding, the computer can be asked to present a summary of all the inferences that can be drawn from the bids made so far. However, the computer appears not to be capable of making hints about the correct bid—if you want the computer to suggest bids, control of the auction has to be passed over to it, and it retains this control, bidding for all players, until the auction is finished. One problem with this program is that it appears to come without documentation, making it difficult to interpret the computer’s bids with certainty, although it appears to adhere to a basic Standard

American system. Also, there is no scope in the system for competitive auctions.

A.3 Bridge Buff

Producer BridgeWare.
Address Toronto, ON, Canada.
Tel (416) 463-2689.
Price \$125 (Can).

A review of version 1.0 of Bridge Buff in Canadian Master Point magazine (June 1992) was rather unflattering, finding the interface to be ‘just not user-friendly’ and the mechanics of bidding ‘cumbersome’. Another major problem was that the program could not be used to ‘actually play bridge’—it could defend against your declarer play, but not play the role of declarer itself.

The ‘solid recommendation’ received by version 2.0 in CMP’s May 1994 review, therefore, represented a large improvement, attributable to an upgrading of almost every area of the program. For example, the basic 2/1 bidding system of the original was augmented to include what CMP described as a ‘list of different conventions and treatments... long enough to give you a good selection of ideas you might want to try out on the computer before using them in a real game’. The new version also featured a declarer play module, and the graphics were also now ‘clean and attractive, especially if you have VGA available.’ CMP describes the standard of play of version 2.0 as follows:

Bridge Buff... will bid and play against you at least as well as any of its competitors. It makes mistakes, but our testing did not catch it out in any of the wildly ludicrous kind of errors to which many bridge programs are occasionally prone.

One issue that is not addressed by CMP’s review is whether the new version of Bridge Buff improves its play by ‘peeking’ at the other hands. This approach helped the earlier version to defend ‘well above the standard of other programs’, but was an integral part of the algorithm which could not be ‘turned off’. The downside of this increase in the computer’s knowledge is that subtle deceptive plays become pointless. Such practices also make a comparison of the program’s playing strength against other Bridge systems rather unfair. Manley’s review [Manley 94] reports that the new version does still peek,

and that this helps it to play at a level slightly stronger than most other programs, with the possible exception of Positronic Bridge.

A.4 Bridge Champion with Omar Sharif

Producer Oxford Softworks
Address Stonefield House, 198 The Hill, Burford, Oxfordshire, England OX8 4HX
Price £39.99

Many options for bidding including Acol, Standard American, or 5 Card Majors, at either beginner or advanced level, which can then be adapted according to taste from a wide range of conventions. However, Forrester [Forrester & Dimbleby 95] describes the level of bidding as ‘not good’, and Christina Erskine of PC Review magazine comments that ‘it’s very willing to bid itself into high-level sacrifices in which it goes down massively’. Card play is described as ‘reasonable’ in [Counsell 94], and by Forrester as ‘passable most of the time, occasionally terrible’. However, this was the only Bridge program which Forrester could bring himself to recommend anyone to buy, with the proviso that any prospective purchaser should be made ‘fully aware of the limitations’.

A.5 Bridge for Windows

Producer Full Circle Computing.
Address 15 Greenridge Avenue, Suite 19, White Plains, NY 10605-1248.
Tel/FAX 914 997-1774.
Price \$49 + \$4 s& h (\$15 outside US/Canada).
Note An evaluation copy is also available for \$6.

This program describes itself as being ‘designed to provide relaxation’. It uses weak two-bids with two clubs as the only forcing bid, but during card play can only take the role of the defenders. To increase the chances of the user becoming declarer the best hand is given to South (the user) and North (user’s partner). During the bidding, East and West’s bids, as well as North’s, are automatically generated.

The following extract from the program’s readme file seems to indicate that Bridge For Windows is an expert-system type application with a collection of rules designed

to recognise varying situations.

An attempt was made to deal with all combinations which may arise, however, there may well be situations, especially in bidding and responding, which may not be dealt with correctly. When you find such situations, please communicate this information to Full Circle Computing...

A.6 Bridge Master

Producer Fred Gitelman, Bridge-Base Inc.
Address 15 Lillian Street, Toronto, Ontario, Canada M4S 2H7
Tel (416) 322-8316
Price US \$59.95. Additional disks of 30 deals: \$14.95.

Bridge Master is an educational tool to instruct and improve the user's declarer play. It received a glowing review from Eric Rodwell in the November 1992 issue of the Canadian Master point magazine, along with further positive reviews elsewhere by Alan Truscott and Brent Manley.

The system is based on a collection of pre-made deals ranging in difficulty from 1 (for novices) to 5 ('extremely challenging even for expert players'), which the user plays in an attempt to make a pre-defined contract. If the line of play selected is not optimal, the computer will always ensure that the defenders' cards fall 'unluckily', thus defeating the contract. For example, if it is possible to guard against a 5-1 break in a side suit, the program will make sure that it breaks 5-1.

When the play of a hand is over, there is the option to view a screen presenting the correct line of play, and the rationale behind it. This portion of the program particularly impressed Rodwell:

As you read through the write-up, the hand diagram on the screen will change to reflect what you are reading about. For example if part of the text says 'If East has four trumps to the jack', then as you read through that part of the text, the East/West hands will change so that East now does have 4 trumps to the Jack. It is like watching a movie, or reading a bridge book without having to worry about crossing out cards or flipping pages. This is an excellent medium for bridge instruction.

However, Tony Forrester, writing in the Daily Telegraph, was not so enthusiastic. He gave the program a score of just one out of a maximum five, and commented, 'The

hands are too abstract to be of practical value. They are like crossword puzzles: fun to do but they are not going to improve your bridge and won't win you any tournaments. If it cost a tenner it might be worth it, but at this price you would be better off buying a book'.

Two versions are available, each with 90 hands: the Basic/Intermediate version with 30 hands from each of the levels 1, 2 and 3, and the Intermediate/Expert version with 30 hands from levels 3, 4, and 5. Extra disks of 30 hands from a particular level can also be purchased, and several well-known Bridge authors have apparently 'shown an interest in putting their books on disks to be used with the Bridge Master program'.

A.7 Bridge Master

Producer Capstone

Confusingly, Capstone also produce a program which bears the name 'Bridge Master', but this namesake offers the more familiar functionality of allowing the user to play random hands against the computer, and runs on the Apple Macintosh rather than the IBM. Capstone's Bridge Master has reasonably attractive features, such as support for play over a modem or network (with the computer filling up to three of the empty seats) and bidding options that include weak or strong 2 opening bids, Gerber or Blackwood, preemptive bids, Stayman, take-out doubles and transfer bids. Unfortunately, however, it is let down by its playing strength, which has led to very unflattering reviews.

A.8 BridgeMate

Producer B. Richardson.

Address 8845 42nd Ave. S.W., Seattle, WA 98136.

Price Shareware. Registration \$49.

Email brmate@hebron.connected.com.

BridgeMate offers a wide variety of bidding options, allowing users to fill out their own convention cards based on Standard American, 2/1, Kaplan-Scheinwold, or Precision. Conventions include a weak or strong No Trumps which can be varied by points count,

seat, or vulnerability, Jacoby, Texas, Smolen, Lebensohl, and other systems over No Trumps openings, New Minor Force, Ogust, Fourth Suit Forcing, and many more.

However, during the play of the hand, BridgeMate ‘peeks’ at the other players’ cards. This is a ‘feature, not a bug’ in the program, and is of course intended to make the computer a more challenging opponent. However, as we remarked earlier, this makes any comparison with other systems which do not have this advantage rather unfair, and also negates the purpose of any deceptive plays.

The demo version of BridgeMate is restricted to playing one of 50 pre-set hands, but with the full version you can deal random or pattern hands or enter your own custom hands.

A.9 Bridge Olympiad

Producer QQP Software.

Price US\$49.95.

The appeal of Bridge Olympiad is the context it can provide for its bridge play: it allows the user to choose a partner from a pool of available players and go to the World Championships, where they play a complete round-robin of 8-board matches against seven other teams which include Italy, USA and China. According to a review in Canadian Master Point [Lee 94] ‘you won’t likely be a winner first time out... and you will keep coming back until you do win’.

However, the actual play of the system is not given a very a flattering review. Its bidding treatment is described as ‘very limited’ and ‘capable of generating extremely bizarre auctions’, and its card play is described as making ‘horrendous errors in play and defence’. Specific play deficiencies highlighted are ‘diabolical’ opening leads such as unsupported honours ‘especially from a short suit against 3 No Trumps (K from Kxx for example)’, and ‘a spade from three small against 6 No Trumps, instead of cashing its AK’. Apparently, when playing the role of declarer it also looks at the defenders’ hands, leading to such strangeness as playing AKQ10xx opposite J7x by leading low to the 7.

In summary then, Bridge Olympiad, may be a good Bridge *game*, but it appears not to be a good Bridge player.

A.10 Bridge Pal

Producer WR Software.
Address P. O. Box 4819, Walnut Creek, CA 94596.
Price Shareware. Registration \$9.95.

The card play of this program is rudimentary at best. For example, in a suit contract with the Q987 of trumps in dummy, a computer defender holding the Jx (the only other trumps remaining in any hand) will duck the lead of the 9! This program also scores rather poorly in the user-friendliness stakes, as it offers no hints during card play, and allows no backtracking over 'mistakes'. Hints are offered during the bidding stage, but must be accepted once requested.

The interface as a whole is also rather visually unappealing and awkwardly constructed (with, for example, the user being queried about changing the program speed setting on repeated occasions during a single session). It also uses different commands to start up in EGA or CGA mode, despite the visual appearance of each mode being wholly identical. It is possible, however, that some of these problems may have been rectified in later releases (our copy dates from 1986).

A.11 Grand Slam

Producers Electronic Arts.
Address P.O. Box 7578, San Mateo, California, CA.
Tel 94403-7578.

The original Grand Slam Bridge has now been superseded by the release of Grand Slam Bridge II. Unfortunately, however, it appears that it is the cosmetic features (such as enhanced graphics and extra options) that have benefited from this revision, and not the play component itself. Many users have complained about the speed of GSBII, with, for example, the reviewer in Canadian Match Point Magazine (March 1993) reporting that it ran 'so slowly that it was unplayable on my 25 Mhz 386, and

I had to borrow my son's 486 to examine the program. Even on the 486 it was much slower than the older version'.

Disappointingly, it appears that the playing algorithms do not benefit from this increase in 'thinking time'. Indeed, the CMP review states that 'the new version does not play or bid any better than the original'. We can speculate that this may be due to way in which the original version was based on a large library of flow charts (the cover blurb on the packaging describes the designers as 'devouring the literature, translating the information they found into flow chart after flow chart'). Resolving the conflicts between the rule-sets resulting from such large collections of flow-charts becomes more difficult as the number grows, and maintenance and development become correspondingly more complicated. The significant reduction in speed of the new version also strongly suggests a look-ahead search algorithm, which may have been introduced to enhance the performance of the original code, or perhaps even as a complete replacement of the play algorithms.

Some play features, such as the bidding, were enhanced by the upgrade, with the addition of Jacoby transfers, gambling 3 No Trumps and unusual 2 No Trumps to the original Standard American bidding options of cue bids, strong or weak 2 bids, and 4 or 5 card majors. The general consensus, however, is that the original version is equally as good as its successor.

A.12 Meadowlark Bridge

Producers Meadowlark Software Co.

According to a review in Canadian Master Point Magazine (Jan 1994), Meadowlark Bridge 'plays at about the level of a bright beginner, so most players will find it an aggravating partner, and an unchallenging opponent'. This view is supported in [Manley 94], where Bob Sweeney reports that the 'play of the cards and defence are prone to unexplainable lapses', and the overall playing skill is given a poor rating compared to other systems.

The package as a whole, however, fares rather better. The CMP review likes the user

interface ('second to none'), the 'extensive list' of bidding conventions, and the hand generator, which is 'very easy to use'. Meadowlark requires Windows to run, but according to the CMP review, makes full use of the potential of this environment to gain its recommendation as 'one of the best bridge packages available'. A demo version can be downloaded from <ftp.rrnet.com>.

A.13 Microbridge

Producer J. M. MacLeod.

Address 11 St. Aubyn's Avenue, London SW19 7BL.

Tel 081-947-9755.

Microbridge is essentially an educational tool which helps users to practise and improve their bidding skills. It has two play modes: lesson and practice, the first concentrating on specific aspects of bidding, and the second allowing the user to bid random hands and then to play the resulting contract.

For beginners, the lesson mode will be the most useful. With it, they will be able to build up their bidding from scratch by following the progression of sixteen lessons contained in the 70-page manual. Each of these lessons focuses on a particular aspect of bidding, such as opening, strong two bids, fit-finding, or slam bidding. The user first reads through the explanations in the manual and then uses the program to access a set of deals tailored to practise the knowledge that has been newly acquired. Correct bids are accepted without question, but incorrect bids are queried by the program, which presents a list of features of the user's hand which do not agree with the interpretation of the bid. As the bidding progresses, valid inferences about the cards held by North and South are presented and maintained on the screen.

More experienced players will probably prefer to use the practice mode, which allows play on random deals, but retains the options of displaying the facts inferable from each bid, and of signalling when there is a discrepancy between the interpretation of their input bid and their actual hand. This time, when the bidding is complete, the user has the option of playing the contract, although hints are no longer available, and the card play is not particularly strong (the manual claims that 'although this part of the system can so to speak be regarded as being "for amusement only", you should

find that your computer opponents will give you a good game').

In addition to not actually being able to play a game of Bridge against the computer (which, after all, is a forgivable fault in a system which sets out its stall as a bidding tutor), there are a few other minor drawbacks to this package. Firstly, there is the lack of diversity in the conventions, with only standard Acol (supplemented by Stayman and Blackwood) on offer, and the only option being a choice between weak or strong No Trumps. Also, there is no provision for competitive bidding. On the interface side, the graphics, although adequate, are only CGA, and the key-disk copy protection is bound to deter some.

However, the blurb on the back of the manual includes recommendations from the Times Higher Education Supplement ('A teaching aid of real value'), and from BRIDGE International ('A highly original and effective bridge tutor').

A.14 Micro Bridge

Producers Tomio Uchida and Yumiko Uchida.
Address 2-1-7 Mizonuma Asaka-shi, Saitama 351, Japan.
Email uchida@rkmath.rikkyo.ac.jp
Price \$60 or 8000 yen.

This is a DOS program with VGA graphics that can function in duplicate mode, pair match mode, or allow two human players to play against the PC at the same time via a serial link. It allows a choice of bidding systems and apparently is reasonably fast, although it is reputed to be weak at slam bidding.

A.15 Micro Bridge Companion

Producers T. Throop, Great Game Products.
Address 8804 Chalon Drive, Bethesda, Maryland.
Phone (301) 365-3297.
Email BRIDGEBARON@mcimail.com.
Price \$59.95.

Version IV of Micro Bridge Companion was reviewed in the January 1992 edition of Canadian Master Point magazine. This review gave the following reasonably positive

summary of the program's playing strength:

Micro Bridge Companion does not play bridge as well as most club duplicate players, but it does play bridge well enough to be enjoyable, most of the time.

However, it also pointed out some deficiencies with the standard of play and the bidding. For example:

It is prone to inviting to game and then going to slam when you accept the invitation... Perhaps the most frustrating moments occur when you are defending with the computer; you can feel very helpless when the computer won't cash the setting trick. You learn never to give up as declarer and I have found some bizarre ways to make impossible contracts.

Version V of the program was released later in 1992, and included a longer list of bidding conventions, such as negative doubles and a forcing response to 5-card majors, where before the only real selections had been weak two bids and the type of jump overcall. This new version was also reviewed by CMP magazine, this time in January 1993. Despite acknowledging that the program could 'make fairly elementary errors in play and defence... even on its highest level setting', this review's overall evaluation was that Micro Bridge Companion was 'probably the best bridge playing package available'. From the perspective of the strength of its play, at least, this assessment is borne out by the facts, since the program contains the code from 'Bridge Baron' — the winner of the Bridge section of several computer Olympiads, and also of a recent three-way contest in America. In the comparative review of the magazine of the American Contract Bridge League [Manley 94], MBC was one of only three programs to score over ten out of 20 for playing skill. Even so, the July issue of Bridge World tested the program to see whether it 'bids, plays or defends better than a beginner'. Their conclusion: 'It doesn't'.

Many of CMP's positive comments about Micro Bridge Companion are directed at the interface and the play options available, which include a library feature for saving and annotating your own games, the ability to play duplicate (with the computer playing all four hands in the 'other room'), and a number of instructive hands taken from Bridge World, which can be played through a card at a time to practise particular problems. There is also a monthly 'Bridge \$ Competition' which can be entered by playing 16

boards against the computer and sending your result to Great Games Products. Access to a list of these monthly scores would be very interesting, since a downward trend would provide an indication of whether revisions of the play algorithms were producing an increase in performance.

A.16 Omar Sharif's Bridge

Producers	CP Software.
Price	£39.99
Tel	0993 823463.

The general opinion of Omar Sharif's Bridge is not too flattering. A review in the July 1992 issue of PC Review [Erskine 92], for instance, described the computer players as 'transparently software generated', and the program's card-play as 'frustrating'. Tony Forrester [Forrester & Dimpleby 95] was also not impressed:

...a lot of the play of the hand focuses around playing trumps because it is an easy thing to program. If it can't think of what to do it will lead a trump. If you have a queen doubleton don't bother trying to hold your cards up, it will always lead Ace, King straight out.

The program has also been witnessed to pass on jump shifts, leave the bidding in five card fits when there is a nine card fit in another suit, and lead its four card suit against a No Trumps contract, despite the opposition bidding the suit.

For bidding, the computer uses either standard Acol or 5 card majors, but this choice must be made when the program is started. Stayman and Blackwood are the only other conventions. There is a line in the manual which reads, 'the computer uses many complex bidding strategies and it may deviate from the rules... in certain situations'. Unfortunately, there is no hint as to what these situations might be, although the reviewer in PC Review claims to have 'detected the odd take-out double and cue bid'.

On the plus side, the package does boast a visually impressive interface. It also utilises Soundblaster support to provide digitised comments from Omar Sharif such as, 'The contract is, Four, Hearts, by, North. East, commences play'. Unfortunately, however, many users quickly tire of hearing 'That was *good* play', and 'Odd luck, try again', every time they win or lose a trick.

A.17 Oxford Bridge 4

Price £49.95

Forrester [Forrester & Dimbleby 95] gives this program a score of one out of a maximum of five, and says 'I wouldn't advise anybody to buy this'. As examples of its poor play he claims that the program will not even double if the user bids wildly to game when the computer holds all the cards, and that during card play 'if it has eight top tricks left and that is its contract, it doesn't go for over tricks even if they are blatant; it just cashes the eight top tricks'. Forrester also complains that 'the graphics are not clear, with information clustered in corners and a lot of space unused'.

A.18 Positronic Bridge

Producers	Positronic Software Inc.
Address	1318 Main Street, Dartmouth, Nova Scotia, Canada B2W 5N7.
Price	Competitor: US\$99.95, Expert: US\$199.95, + \$5 s&h.
Email	mlantz@fox.nstn.ns.ca.
Tel	(902) 434-6444, or 1-800-565-4005.
FAX	(902) 435-6792.
Distributor	Readysoft of Canada (416) 731-4175.

Positronic Bridge comes in two flavours: competitor and expert. Both these versions have the same play routines, but the expert version incorporates a GA-based algorithm which allows the user to 'teach' the software to bid in the way that they want it to. The GA component was developed by Dr. David Lever, Ph.D., Dalhousie University, but unfortunately, since the program is commercial, there are no papers describing his approach. Positronic Bridge was reviewed in June 93 Bulletin of the ACBL by Brent Manley, and in the November 1993 Canadian Master Point (CMP) Magazine by Ray Lee. This latter review summarised the program as follows:

As a bridge partner or opponent, it is no worse than the competition... in other words it plays at an intelligent novice level. It defends mechanically, and does not play the hand very well, although according to the manual, the learning routines should enable it to improve in both these areas over a period of time.

As the review points out, this ability to learn may not be of particular benefit to non-expert users, since the improvements are gleaned from analysing games played against the user. It also seems curious that the standard of play is not initially rather stronger: if the learning algorithms were truly effective it might be expected that the developers would have invested some time in playing the system against expert players in order to produce a higher level of performance. In [Manley 94], where the program was tested by Life Masters, the program's playing skill was indeed rated higher than any other program, but not significantly so.

Both versions of the program feature a wide collection of bidding conventions based on Standard American and including, for example, cue bids, splinters, forcing and non-forcing Stayman, Lebensohl and Roman Key Card Blackwood. In addition, the Expert version allows you to enter 'teaching' mode which brings up a display of a complete deal and allows you to 'correct' the computer's bids for each hand until they begin to reflect your personal preferences. This aspect of Bridge is clearly the best suited to a learning algorithm since it bypasses the 'credit assignment' problem (of deciding which move(s) from a long sequence are responsible for good or bad performance) by requiring the user to assess each individual bid. The CMP review, however, found the learning to be 'painfully slow', with, for example, an attempt to teach the program to open 1 No Trumps on 15-17 point hands taking four or five hours of work to produce the correct bid 'most of the time, but still not on every hand'. Although the magazine later received an updated version of the program that 'seemed to learn much more quickly', they still suggest that it will 'require a heavy time investment to teach it an entire system'. Many of the complaints about this program have been directed at the copy protection—a now old-fashioned disk-lock scheme which requires the master diskette to be present in the drive throughout play. This aside, however, the interface is reportedly user-friendly and 'attractive'.

The competitor version can be upgraded to the expert version by the purchase of an upgrade module, and there is a 30-day money back guarantee.

A.19 Saitek Industries

Producers	Saitek Industries Ltd.
Address	4 Bridge Studios 318-326, Wandsworth, Bridge Road, London SW6 2TZ.
Tel	071-736-7596.
Fax	071-731-7684.
Address	Suite 101, 2291 W 205th St. Torrance, CA.
Tel	213 212-5412.
Price	Pro Bridge 510: £200, upgrade chip: £50.

Saitek make three dedicated Bridge computers. The Bridge Shadow is a hand-held product about six inches long, three inches wide and an inch tall. It has an LCD screen and runs off AAA batteries. It bids Standard American 5-card majors, strong No Trumps, strong two-bids, Stayman and Blackwood.

The features of the Bridge Shadow are built upon by the Pro Bridge 310, which plays ACOL in addition to Standard American, but the top end of Saitek's Bridge computer range is the Pro Bridge 510. This is also battery-operated, but is no longer small enough to be hand-held, being closer in size to a lap-top computer. The Pro Bridge 510 was reviewed in the magazine of the Scottish Bridge Union [Sime 92], which although acknowledging that it was 'reckoned to be the best dedicated (*i.e.*, solely Bridge) Bridge computer on the market' identified numerous shortcomings:

...things which it won't do are:-

- Refuse a trick, except in straightforward hold-up situations.
- Take any notice of your signals or discards.
- Play a false-card or be taken in by one.
- Be squeezed or end-played (it prefers to take evasive action by discarding foolishly).

Tony Forrester [Forrester & Dimbleby 95] was also not impressed, writing that 'The play is poor. In defence, it played clubs round to king, jack, dummy, rather than in spades where dummy had a void'. Forrester also found the screen hard to concentrate on, comparing it to 'early hand-held Gameboys, with minuscule liquid crystal icons'.

The 510 model increases the variety of bidding systems over the 310, this time including Precision. However, Forrester describes the bidding as 'basic' and the SBU review

claims that the computer has a disconcerting tendency to pass forcing bids and also reports that 'in the doubling zone it is fairly eccentric. If it doubles it usually means there is a play for the contract'.

There is a 'computer peeks' option which helps to improve both the bidding and the play. However, setting this option is described as improving the program's defence 'from abysmal to poor' — a weakness for which it 'compensates, and prevents over-bidding, by arranging for most finesses to fail'.

There is an upgrade chip available which may address some of the idiosyncrasies described above. The machine also has many extra features and optional extras, including an 'add-on' screen that allows another human to play as your partner.

Appendix B

Generating Explanations

Tell me more.

— ELIZA
The program

This appendix looks at the way textual explanations are generated from expressions in FINESSE's uncertainty representation language. §B.2 describes the basic approach, whilst §B.3 outlines a number of ways in which the results can be improved, and proposes a more effective procedure derived from genuine natural language systems.

B.1 Review

Recall that FINESSE uses the interpreter algorithm of Chapter 7 to identify promising lines of play in single suits. Rather than using bare probabilities to describe the possible returns from each branch, this algorithm utilises an uncertainty representation language: disjunctions of terms we call \mathcal{C} -conjunctions are used to represent the conditions under which various numbers of tricks will be produced. For a suit in which there are p sequences of outstanding critical cards and one sequence of low cards, a \mathcal{C} -conjunction will be of the form:

$$\text{east}(\text{Bin}_1, \text{List}_1) \wedge \cdots \wedge \text{east}(\text{Bin}_p, \text{List}_p) \wedge \text{east_low}(\text{Bin}_{p+1}), \quad (\text{B.1})$$

where each Bin_i is a binary string describing the possible cards that can be held by East from the corresponding card List_i (or from the low cards, if $i = p + 1$).

B.2 A Basic Solution

FINESSE's basic approach to generating explanations for such expressions is to produce a single piece of text for each `east` or `east_low` term. Examples of such explanations are 'East holds the King', or 'West holds any 3 of the remaining 7 low cards'. The explanations for complete \mathcal{C} -conjunctions are then formed by linking such individual explanations into phrases by appropriate use of the word '*and*', and then forming sentences by using the word '*or*' to connect these phrases together.

We first describe the basic pattern matching process for generating these explanations, and then in §B.3 discuss ways in which the output of this basic procedure can be made slightly more natural.

B.2.1 Pattern Matching

The textual explanation describing the distribution of a *single* sequence of outstanding cards (*i.e.*, the distribution represented by the binary string of a *single* `east` or `east_low` term) consists of four components, each based on one of the following factors:

- **Viewpoint.** The purpose of any explanation will be to describe how the cards concerned may be shared between the two defenders. Clearly, since the cards held by one cannot be held by the other, it is wasteful to describe *both* defenders allocations. A 'viewpoint' is therefore selected and the sequence described with a phrase beginning 'East holds ...' or 'West holds ...'.
- **Quantity.** Once a viewpoint has been selected, some text is required to describe *how many* cards the chosen player's hand contains in the sequence. Here, attention is given to special cases, such as the use of 'both' in explaining sequences of just two cards.
- **The actual cards.** The actual cards represented by the sequence also have to be described. For example the sequence may consist of the 'Ace, King and Queen' or simply represent the 'remaining low cards'.
- **Other.** The system's output can be made slightly more natural by checking for the special case where a sequence of cards is split evenly between both defenders.

When this is the case, the components describing the viewpoint and quantity of the cards held by each player are replaced by a component which describes the sequence as being ‘split evenly between East and West’.

Figure B.1 shows how these four components can be formed into a phrase. Any combination of selections from the first three columns forms an allowable explanation. Selecting a component from the final column is only allowed if no component is selected from the first two columns.

Viewpoint	Quantity	Cards	Other
East	holds both	the Card₁ , Card₂ , ... and Card_i	
West	holds	the remaining low cards	
Either East or West	holds any n of		
	holds at least n of		
	holds n₁ , n₂ , ... or n_i of		are split evenly between East and West

Figure B.1: Canned text for describing *C*-conjunctions

Given a term of the form `east(Bin,List)` or `east_low(Bin)`, the first task is to select the defender from whose viewpoint to generate the explanation. This choice would ideally take into account information such as the danger hand, or the perspective from which other sequences in the same sentence were being explained. *FINESSE*, however, simply chooses a new viewpoint for each `east` and `east_low` term in isolation, selecting each time the player whose longest possible holding from the sequence concerned is greatest (since this player’s holding is assumed to be the more ‘important’). If West is the player selected, the value of `Bin` is then modified to reverse its binary representation. (Recall that bit *i* in the binary string represented by `Bin` is set to 1 if East may hold *i* cards in the given sequence. In this situation, West must hold the remaining $N - i$ cards, so West’s possible holdings can be found by simply reversing the string describing East’s cards. §B.4.1 describes the procedure used to reverse binary strings.) The text describing the quantity of cards held is then determined by matching `Bin` against the possible patterns of Figure B.2 on the following page

Binary String	Text generated
string length 3, only first bit set to 1 (<i>i.e.</i> , 100)	holds both
string with only first bit set to 1	holds
string with just one bit (bit n) set to one	holds any n of
string with all 1 bits occurring consecutively at start ¹	holds at least n of
string with i bits (bits n_1, n_2, \dots , and n_i) set to 1	holds n_1, n_2, \dots or n_i of

Figure B.2: Patterns matching canned text for quantity of cards held

The component describing the outstanding cards is easy to produce by simply inserting an ‘and’ between the names of each card in an outstanding sequence of high cards, or by just describing low cards as ‘the remaining low cards’. Finally, the component ‘are split evenly between East and West’ is only used for values of `Bin` that are of odd length and have just the middle bit set to one.

As an example, Figure B.3 gives the explanations generated by `FINESSE` for all the possible values of `Bin` in the term `east(Bin, [ace, king])`, whilst Figure B.4 shows the possible results for a term describing a sequence of three low cards. Note that in the cases where `Bin` represents a complete string of 1s, no text is produced, since such terms make no restrictions on the positions of the cards they describe.

Binary string	Explanation generated
001	West holds both the Ace and King
010	the Ace and King are split evenly between East and West
011	West holds at least one of the Ace and King
100	East holds both the Ace and King
101	either East or West holds both the Ace and King
110	East holds at least one of the Ace and King
111	–

Figure B.3: Possible explanations for an outstanding sequence of Ace and King

¹ §B.4.2 describes how a check for such an occurrence is easily made.

Binary string	Explanation generated
0001	West holds the remaining three low cards
0010	West holds any two of the remaining three low cards
0011	West holds at least two of the remaining three low cards
0100	East holds any two of the remaining three low cards
0101	West holds one or three of the remaining three low cards
0110	either East or West holds any two of the remaining three cards
0111	West holds at least one of the remaining three low cards
1000	East holds the remaining three low cards
1001	either East or West holds the remaining three low cards
1010	East holds one or three of the remaining three low cards
1011	West holds zero, two, or three of the remaining three low cards
1100	East holds at least two of the remaining three low cards
1101	East holds zero, two, or three of the remaining three cards
1110	East holds at least one of the remaining three low cards
1111	–

Figure B.4: Possible explanations for an outstanding sequence of 3 low cards

B.3 Further Improvements

By itself, the scheme outlined above has the tendency to produce rather lengthy explanations which, although readily understandable, are rather unlike the explanations typically offered by human players. This is mostly due to the way in which explanations are generated by considering each `east` and `east_low` term separately. The quality of the explanations can be quickly improved by recognising and explaining more naturally a number of patterns which occur on a larger scale either within single \mathcal{C} -conjunctions or between multiple \mathcal{C} -conjunctions. This section describes a few of these special cases which FINESSE can recognise.

B.3.1 Voids, Singletons and Doubletons

Special terminology is often attached to situations in which one defender holds all the outstanding low cards in a suit. This is because in such situations it is usually easier to describe the possible distributions by simply stating how many high cards are held by the other defender. A simple example of this is when one defender holds all the remaining high cards as well as the low cards. Rather than describing this situation

by stating what all these cards are, it is obviously more economical to simply explain that the other defender is *void* (i.e., has no cards) in the suit.

Other commonly used expressions are the terms *singleton* and *doubleton*. Respectively, these describe situations where a player has a total of exactly one or exactly two cards in their hand. For example, a player holding just a King and no low cards would be described as having a ‘singleton King’, and a holding of just an Ace and a Jack would be described as a ‘doubleton Ace, Jack’.

FINESSE uses these terms to simplify its explanations, by treating specially the cases where one player holds no low cards and none of the outstanding sequences of high cards are split between the two defenders. This situation can be simply explained without the use of disjunction. For example, FINESSE translates $\text{east}(000, [k, q]) \wedge \text{east_low}(100000)$ to ‘West holds the doubleton King, Queen’, whereas $\text{east}(010, [k, q]) \wedge \text{east_low}(100000)$ would have to be translated to ‘West holds the singleton King *or* Queen’. The latter of these expressions could be produced as a special case, but relying on a series of *ad hoc* modifications would make the language component hard to maintain. In §B.3.4 we will examine a better technique for generating explanations which handles such cases more naturally.

B.3.2 Symmetry Between \mathcal{C} -conjunctions

There are two cases in which checking for possible symmetry in a formula can lead to improved explanations. The first of these is when separate \mathcal{C} -conjunctions disjoined together in an expression describe distributions which are the opposites of each other (i.e., swapping East’s and West’s cards in the situations described by the first \mathcal{C} -conjunction produces the situations described by the second). Rather than producing an explanation for one disjunct of the form ‘East holds ..., and East holds..., and West holds...’, and then another of the form ‘West holds ..., and West holds..., and East holds...’ FINESSE produces the more natural rendition of ‘Either East or West holds ..., and ..., whilst the other defender holds ...’.

B.3.3 Symmetry Within \mathcal{C} -conjunctions

The second case where symmetry can be utilised is when all the binary strings in a single conjunct are symmetrical about the centre. For example, the explanation of the \mathcal{C} -conjunction

$$\text{east}(010, [\text{ace}, \text{king}]) \wedge \text{east_low}(100001)$$

would be rendered by the pattern-matching of the previous section as ‘If the King and Queen are split evenly between East and West, and either East or West holds the remaining five low cards’. In fact, this situation is a more elaborate version of the special case in §B.3.1 in which *either* player has a singleton, and the following explanation is equivalent and more compact: ‘Either East or West holds the singleton King or Queen’. A simple check for these situations is to look for \mathcal{C} -conjunctions in which each binary string is equal to its reflection. Such \mathcal{C} -conjunctions are explained from the perspective of just one defender and the explanation attributed to ‘either East or West’.

B.3.4 Intermediate Logical Forms

The way in which FINESSE produces its explanations by directly producing a piece of text for each term in a \mathcal{C} -conjunction is essentially incompatible with the need to check for the special cases listed above. Handling such cases correctly would require a global analysis and modification of the entire original expression and of each constituent \mathcal{C} -conjunction before passing the expression to the pattern matcher.

In genuine natural language systems, more flexibility is produced by the introduction of *intermediate logical forms*. These are stages of representation through which an expression will pass as it is processed from the internal logical language into the *surface structure* from which the textual explanation can be directly produced.

An example of this is provided by Davey’s PROTEUS system [Davey 78], which produces a commentary on a game of noughts and crosses. PROTEUS takes as input a list in which each element is a pair describing the player and the square taken by that player.

For example, the first three moves in a game between PROTEUS and a user, ACD, might be:

[[<Proteus> 7] [<ACD> 5] [<Proteus> 3]] .

PROTEUS transforms its input into an intermediate representation, using the significance of each move to judge the number of moves to describe in one sentence, and incorporating information required for building the sentence (*e.g.*, punctuations and conjunctions). The above example is transformed by PROTEUS into the following form:

[[<Proteus> <square 7> start <game> take <square 7>],
 [<ACD> <square 5> take < square 5>], and
 [<Proteus> <square 3> take <square 3>]] .

The first part of this sentence is then realised as:

I started the game by taking the corner...

Although Davey's concerns centre on natural language problems which are not an issue in our system (such as generating pronominal and non-pronominal referring expressions), it would still be possible to learn from this use of intermediate representations in the FINESSE system.

For example, rather than converting each *east* term and *east.low* term directly into text as described in §B.2, we could instead generate an intermediate term of the form *text(Player, Bin, Cards)*. Here, *Player* would be the player from whose viewpoint we wish to explain the constraint (*i.e.*, *east*, *west*, or *east/west*), *Cards* would be either the list of critical cards being described or the atom *low*, and *Bin* would be the original binary string. A typical disjunction of *C*-conjunctions would be converted by this process into a list of the form:

[[text(Player, Bin, Cards), text(Player, Bin, Cards), ...],

 [text(Player, Bin, Cards), text(Player, Bin, Cards), ...]] ,

where each entry would correspond to one of the original \mathcal{C} -conjunctions.

The special cases of the preceding three sections could now be accounted for by processing this list. For example, checking for voids, singletons and doubletons could be achieved by examining each entry of the list, searching for cases where one **Player** only holds complete sequences of critical cards. Thus, an entry in which one term is of the form `text(Player, 2n, List)`, and the remaining terms allocate all the cards to the other defender could be translated into a term of the form `[text(Player, only(n), List)]`. Extending this modification to cover cases where the only cards held by a player come from more than one outstanding sequence is trivial, and with a little more effort it should also be possible to incorporate the cases involving disjunction in this way. The special cases from §B.3.3 can also be incorporated by extending the above modification to search for matches on `text` terms with **Player** set to `east/west`.

Finally, the entire expression could be analysed to look for special cases of the type in §B.3.2. Any two entries whose `text/3` terms differ in just their **Player** assignments could be combined into a single term in which the **Players** are either `east/west` or `west/east`.

The final explanation would be produced by pattern matching each of the entries in the final list (*e.g.*, using same form of information as depicted in Figure B.1).

B.4 Technical Supplement

This section gives some detail on the techniques used to manipulate binary strings within the FINESSE system. It is included to make otherwise obscure pieces of code more understandable.

When considering a group of N outstanding (critical or low) cards, the binary string that describes how many of the cards East may hold will be of length $(N + 1)$ bits. The two procedures described here deal with methods of manipulating and interpreting these ranges.

B.4.1 Reversing a Binary String

We have already seen that the string represented by `Bin` may need to be reversed if an explanation of the form ‘West holds...’ is to be given. Rather than examining each bit in turn and making $N + 1$ recursive steps through the string, we use the scheme outlined in Figure B.5. The middle bit (if one exists) will always remain the same, and the left and right substrings are reversed recursively before being shifted to the right or left respectively. This procedure often results in an efficiency improvement, as substrings consisting of all 1’s or all 0’s require no recursive sub-calls to process.

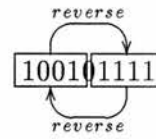


Figure B.5: Scheme for reversing a binary string

B.4.2 When a Defender Holds ‘at least n cards’

After deciding on the defender from whose perspective to explain the distribution, a check is carried out for the case where the string represented by `Bin` allows the player to hold ‘at least n ’ of the cards concerned.

In such a situation, the binary string represented by `Bin` would be of the form $111 \dots 000$ (i.e., an unbroken sequence of 1’s followed by an unbroken sequence of 0’s). In order to detect this situation, we therefore look for some i such that:

$$\begin{aligned}
 \text{Bin} &= 2^N + 2^{N-1} + 2^{N-2} + \dots + 2^{N-i+1} \\
 &= 2^N \left(\frac{1 - (\frac{1}{2})^i}{\frac{1}{2}} \right) \quad (\text{by formula for the sum of a GP}) \\
 &= 2^{N+1} \left(\frac{2^i - 1}{2^i} \right).
 \end{aligned}$$

The rather obscure looking piece of code which checks for this situation is therefore attempting to find the lowest value of i for which $\text{Bin}/2^{N+1}$ is equal to $(2^i - 1)/2^i$. If this equality can be satisfied, then the player from whose perspective we are explaining

the holding must have at least $N - i + 1$ of the cards concerned.

B.5 Summary

We have demonstrated that the uncertainty representation language used by FINESSE is capable of supporting the generation of realistic explanations, and described both the current mechanism for producing such explanations, as well as suggestions for further improvements. We envisage that this capability could eventually be used by FINESSE to explain the rationale behind its choice of actions at any stage of the play of a contract.

Appendix C

Further Examples

How can I lose to such an idiot?

— CHESSMASTER AARON NIMZOVICH
Shouted at the end of a match

This appendix presents some more detailed and extensive examples of FINESSE solving further problems. Specifically, we compare the output of the system against the set of solutions to single-suit card combinations found in the Bridge Encyclopedia [ACBL 94]. §C.1 gives a summary of the system's overall performance on this data set, and §C.2 then presents a series of detailed individual examples.

C.1 Summary of Performance

The Bridge Encyclopedia contains a 55-page section presenting optimal lines of play for a selection of 664 single-suit problems. Of these, a total of 650 examples give solutions which are effectively pure strategies for the maximum number of possible tricks.¹ The fourteen examples which do not give such solutions split into three categories: six problems which give no line of play for the maximum number of tricks, four problems involving the assumption of a mixed strategy defence, and three for which the solution relies on assumptions about the defenders not false-carding.

FINESSE's ability to produce optimal lines of play was tested against the basic 650 problems of the Bridge Encyclopedia, omitting the extra 14 examples which are not

¹ Optimal lines of play for guaranteeing a number of tricks *less* than the maximum are also often included.

comparable. In each case, FINESSE’s line of play for the maximum possible number of tricks was compared to that of the Encyclopedia, producing the results summarised in Figure C.1.

	With non-locality heuristic	Without non-locality heuristic
Correctly analysed	489 (75.2%)	432 (66.4%)
Incorrectly analysed	161 (24.8%)	218 (35.6%)

Figure C.1: Overall FINESSE performance on Bridge Encyclopedia problems

These results reveal a significant amount of non-locality, and also demonstrate the utility of the non-locality heuristic introduced in Chapter 7. This heuristic corrected the analysis of 57 problems whilst not disturbing the analysis of any of those which were already correct. Of the remaining incorrect problems, FINESSE either selects (due to non-locality) a line of play with a sub-optimal chance of producing the maximum number of tricks (149 cases), or a line of play which cannot produce the maximum number of tricks under any circumstances (12 cases). On average, the sub-optimal lines of play miss the maximum number of tricks with a probability of 0.07. Thus, if the entire problem set were to be played with randomly distributed outstanding cards, the expected number of cases where FINESSE would fail to obtain the maximum number of tricks would be 11 (approximately 1.7%).

In terms of speed, the current version of FINESSE is rather slow on this problem set, requiring on average 11.1 seconds to form a plan, and then a further 93.3 seconds to identify the best line of play within that plan (timings produced on a SPARCstation ELC). However, there are two explanations for this. Firstly, the problem set is a rather hard one, since it is designed as an expert reference: there are no trivial problems (*e.g.*, where declarer has only winners, or where no tricks can be won) so that all the problems at least allow the opportunity for some element of manoeuvre. Second, and more importantly, the main slowdown occurs in the interpreter algorithm, which has yet

to be optimised for speed because it is a relatively recent, development version that incorporates the non-locality heuristic. Previous versions of the interpreter algorithm have operated at almost exactly the same speed as the planning algorithm itself, and we envisage no theoretical problems in returning the current algorithm to this level of performance, largely because its present poor performance is mostly due to the inefficient memory management that results from producing an annotated plan that fails to maintain the original plan's construction of using pointers to repeated substructures.

C.2 Individual Examples

The following pages present a series of examples of FINESSE's performance on selected problems from the Bridge Encyclopedia. It is hoped that these examples will enable even those without the resources to run FINESSE directly to gain an understanding of the system's abilities.

Problem 1

(Solutions agree)

A K Q J 9

N
W E
S

2

Encyclopedia No.	1
Maximum tricks	5
Critical cards	Ten
Applicable tactics	Type 1 finesse of 9 against West Cash Ace
FINESSE prob	0.719
Encyclopedia prob	0.72

FINESSE Explanation:

Cash the Ace. This leads to 5 tricks if either East or West holds the ten and the other defender holds at least three of the remaining six low cards.

Encyclopedia Explanation:

Cash top honors in the hope of dropping the ten

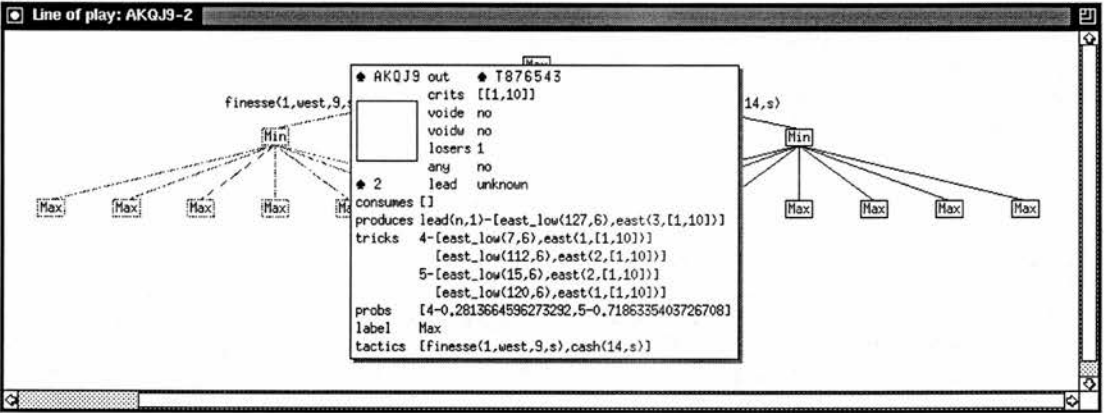
Comments

The comparison between FINESSE and the Bridge Encyclopedia solution on this, the Encyclopedia’s first problem, is fairly typical. The line of play selected by FINESSE is the same as that of the Encyclopedia, but this equivalence is not made completely transparent by FINESSE’s explanation.

Some of the differences between FINESSE’s and the Encyclopedia’s explanations are trivial. For instance, the Bridge Encyclopedia typically gives probabilities only to two decimal places (unless very small chances are being considered), whereas FINESSE works with Prolog’s floating point arithmetic. Also, FINESSE’s explanation generation facility can already be seen to be a little cumbersome. A more compact equivalent to FINESSE’s attempt might be ‘This leads to five tricks if either East or West holds the ten and at most three low cards’. We noted in Appendix B that the best way to improve FINESSE’s explanations would be to replace the current pattern-matching algorithm with an alternative that first translates the basic *C*-conjunctions into an intermediate logical form. Such a system should significantly improve on the explanations found in this appendix.

A slightly more significant difference is FINESSE’s current inability to explain its entire line of play. Instead, it simply describes its intentions by naming the first selected tactic

in its search space (in this case, the cashing of the Ace). The remainder of FINESSE's line of play can only currently be examined using the graphical interface, which for the current problem produces the tree shown below. An interesting avenue for further work on the system would be to add the ability to collapse any given line of play into a tree containing just the distinct possible orderings of its constituent tactics, and then further adding the ability to describe this tree succinctly. In the current example, it should be relatively easy to generate a phrase such as 'Cash the top honours', but in other situations, where different tactics are used in response to different card plays by the defenders, more care would be needed.



Problem 2 (Solutions agree)

A K Q 10 9

N		
W		E
S		

x

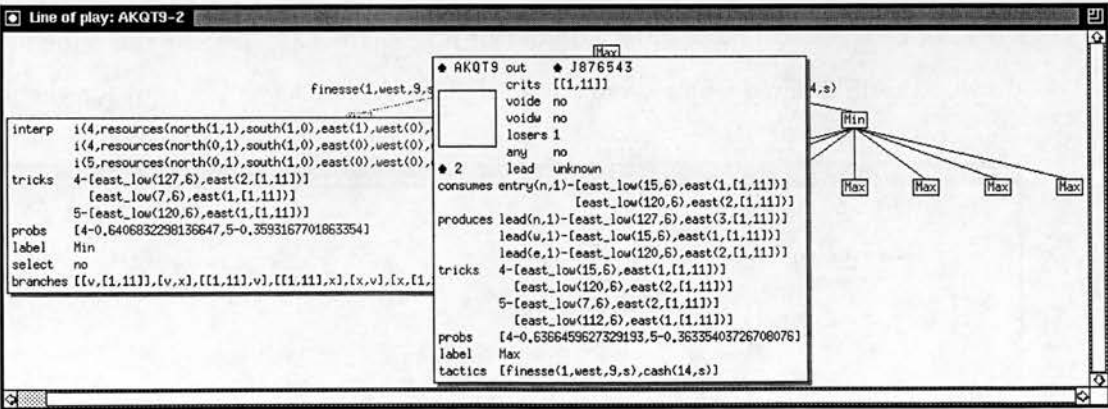
Encyclopedia No.	9
Maximum tricks	5
Critical cards	Jack
Applicable tactics	Type 1 finesse of 9 against West Cash Ace
FINESSE prob	0.363
Encyclopedia prob	0.36

FINESSE Explanation:
Cash the Ace. This leads to 5 tricks if either East or West holds the Jack and the other defender holds at least four of the remaining six low cards.

Encyclopedia Explanation:
Play off the top honors. This is fractionally better than the immediate finesse.

Comments

This example highlights the amount of qualitative information contained in FINESSE’s plans. In the screen capture below we have opened pop-up windows at two nodes. The upper (right-hand) window describes the optimal line of play of cashing the top masters, hoping for the Jack to drop. The lower window describes the alternative line of finessing the nine, which can be seen to have the slightly lower probability of 0.359. Thus, we can quickly demonstrate the Encyclopedia’s claim that cashing is ‘fractionally better than the immediate finesse’.



Problem 3 (Solutions agree)



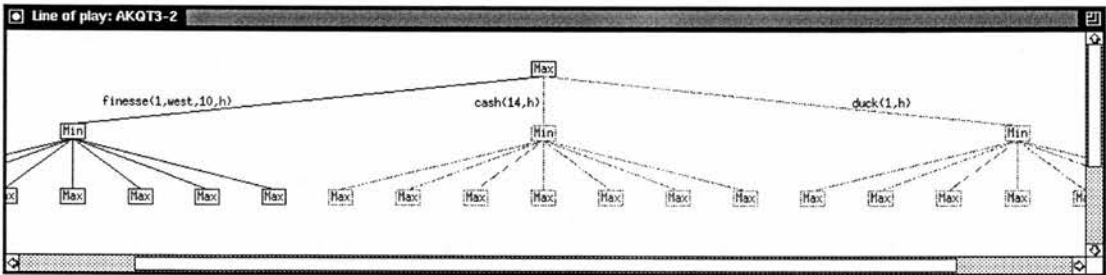
Encyclopedia No.	10
Maximum tricks	5
Critical cards	Jack
Applicable tactics	Type 1 finesse of ten against West Cash Ace Duck a round
FINESSE prob	0.311
Encyclopedia prob	0.31

FINESSE Explanation:
Finesse the ten (Type 1). This leads to 5 tricks if West holds the Jack and East holds three or four of the remaining six low cards.

Encyclopedia Explanation:
Finesse the ten.

Comments

Replacing North's nine with an arbitrary low card in the previous example results in the best line of play switching from the cash to the finesse. FINESSE correctly identifies this, preferring the finesse tactic over the cash tactic (which has a probability of 0.266 of producing 5 tricks). Notice from the screen capture that when more than two tactics are applicable in the initial state, the overall plan may become too large to fit comfortably into a single window. When this is the case, the entire tree can still be examined using the interface's ability to scroll the display window, to limit the drawing of trees to just a small number of levels (the examples in this appendix just display the first three levels), and to select different nodes as the root for the display.



Problem 4 (Solutions agree)



Encyclopedia No.	12
Maximum tricks	4
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of 9 against West Cash Ace
FINESSE prob	0.24
Encyclopedia prob	0.24

FINESSE Explanation:

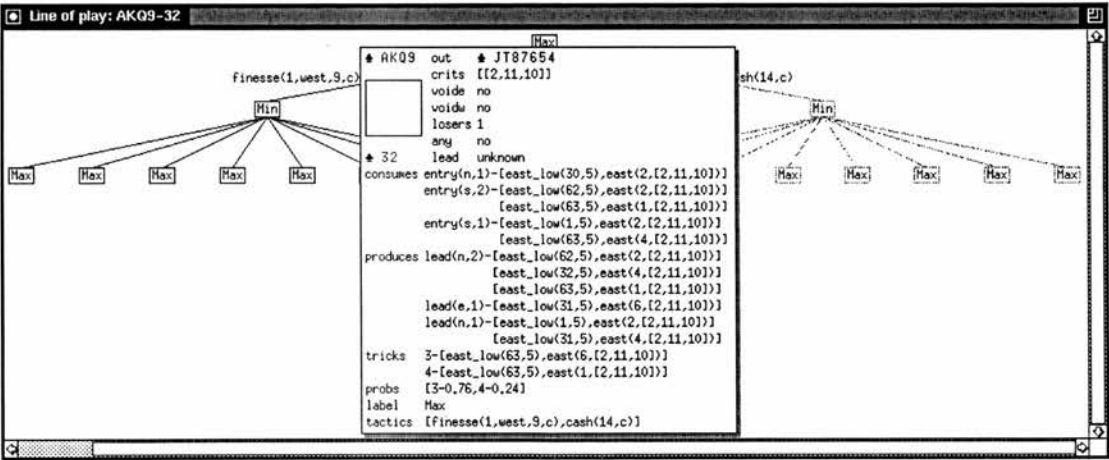
Finesse the nine (Type 1). This leads to 4 tricks if West holds both the Jack and ten.

Encyclopedia Explanation:

Finesse the nine; hope that West has both the jack and ten.

Comments

Note the close correlation between FINESSE’s and the Encyclopedia’s explanations for this problem. Note also that this problem was an example of a situation whose solution was corrected by the incorporation of the non-locality heuristic. Without the heuristic, FINESSE calculates that the finesse of the nine has a probability of just 0.054 of producing four tricks. This results in the cash tactic being selected ahead of the finesse, since this has apparently more favourable probability of 0.061. We will give a more detailed example of non-locality in the following problem.



Problem 5 (FINESSE is incorrect)



Encyclopedia No.	19
Maximum tricks	5
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of 9 against West Cash Ace Duck a round
FINESSE prob	0.300
Encyclopedia prob	0.39

FINESSE Explanation:
Cash the Ace. This leads to 5 tricks if the Jack and ten are split evenly between East and West and the remaining four low cards are split evenly between East and West, or if West holds both the Jack and ten and East holds at least three of the remaining four low cards.

Encyclopedia Explanation:
Play off the top honors, hoping that the jack and ten drop in three rounds.

Comments

This example gives a flavour of the types of non-locality that can arise. If we approach this combination by cashing the Ace (the correct play) and see East play the Jack, what is the best continuation? The table below shows the possible worlds under which finessing the 9 or cashing the King would produce five tricks.

Split	Worlds in which finesse succeeds	Worlds in which cash succeeds
6-0	—	—
5-1	—	—
4-2	Txxx-Jx, Jxxx-Tx	—
3-3	Jxx-Txx, Txx-Jxx	Jxx-Txx, Txx-Jxx, xxx-JTx
2-4	Jx-Txxx, Tx-Jxxx	Jx-Txx, Tx-Jxxx
1-5	—	—
0-6	—	—

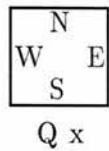
The probabilities of these distributions actually occurring for the finesse and the cash respectively are 0.472 and 0.430. However, the actual best line of play is the cash — the probability of success for the finesse is inflated by including the distributions

where East holds a doubleton Tx or Jx. In an actual game, East would not contribute a critical card from such a holding if he thought that declarer's strategy was to play the finesse on the next round. Since the best defence model of a game assumes that MIN chooses his strategy after MAX, these world states should therefore be ruled out if we select the finesse. FINESSE does not make this inference and therefore makes the wrong selection at this internal node. It is the removal of these worlds by the process of subsumption at a higher node (simulating the best defence concept that the defenders would direct the play down a different branch in these worlds) that results in a probability lower than the correct answer.

Note that the play of the Jack from Jx cannot be characterised as a 'bad play' in general, so it cannot be ruled out in all cases. Attempts to do so result in sub-optimal solutions to other situations.

Problem 6 (Solutions agree)

A K 10 x x



Encyclopedia No.	21
Maximum tricks	5
Critical cards	Jack
Applicable tactics	Type 1 finesse of 10 against West Cash Ace Cash Queen Duck a round
FINESSE prob	0.517
Encyclopedia prob	0.52

FINESSE Explanation:
Cash the Queen. This leads to 5 tricks if either East or West holds the Jack and the other defender holds three or four of the remaining five low cards.

Encyclopedia Explanation:
Play off the top honors

Comments

In this example, previously discussed in Chapter 11, FINESSE produces two distinct lines of play for playing off the top honours: one beginning with the cashing of the Queen and the other beginning with the cashing of the Ace. Screen captures of pop-up windows describing these lines of play are shown below (with the line of play starting with the Queen on the left). These lines both have the same chance of producing 5 tricks, but FINESSE chooses to start with the Queen because it has a higher probability of producing four tricks if the attempt to win five fails. This is because if East shows

AKT43 out J98765

crits [[1,11]]

voids no

voidw no

losers 2

any no

Q2 lead unknown

consumes entry(n,1)-[east_low(4,5),east(1,1,11)]

produces lead(n,1)-[east_low(6,5),east(3,1,11)]

lead(s,1)-[east_low(6,5),east(3,1,11)]

lead(e,1)-[east_low(6,5),east(2,1,11)]

lead(w,1)-[east_low(6,5),east(1,1,11)]

tricks 3-[east_low(4,5),east(2,1,11)]

east_low(2,5),east(1,1,11)]

4-[east_low(9,5),east(2,1,11)]

east_low(3,5),east(1,1,11)]

5-[east_low(6,5),east(2,1,11)]

east_low(2,4,5),east(1,1,11)]

probs {3-0,12857142857142856,4-0,3546583850931677,5-0,5167701863354037}

label Max

tactics [finesse(1,west,10,s),cash(14,s),cash(12,s),duck(1,s)]

interp

i(3,resources(north(1,2),south(0,1),east(0),west(1),ew(0)))-[east_low(3,5),east(1,1,11)]

i(3,resources(north(1,2),south(0,1),east(1),west(0),ew(0)))-[east_low(4,5),east(2,1,11)]

i(4,resources(north(2,2),south(0,1),east(0),west(1),ew(0)))-[east_low(4,5),east(1,1,11)]

i(4,resources(north(2,2),south(0,1),east(1),west(0),ew(0)))-[east_low(8,5),east(2,1,11)]

i(4,resources(north(1,2),south(0,1),east(0),west(0),ew(0)))-[east_low(1,5),east(2,1,11)]

i(5,resources(north(1,2),south(0,1),east(0),west(0),ew(0)))-[east_low(3,5),east(1,1,11)]

tricks 3-[east_low(3,5),east(1,1,11)]

east_low(4,5),east(2,1,11)]

4-[east_low(9,5),east(2,1,11)]

east_low(3,5),east(1,1,11)]

5-[east_low(6,5),east(2,1,11)]

east_low(2,4,5),east(1,1,11)]

probs {3-0,1350248447204963,4-0,3472043689440936,5-0,5167701863354037}

label Min

select no

branches [(v,1,11)],(w,x),[(1,11),v],[(1,11),x],(x,w),[(x,1,11)],(x,x)]

out when the Queen is cashed, a finesse of the ten can still be carried out on the second round. The chance of East being void is 0.07, which is the discrepancy between the probabilities of gaining four tricks in the profiles of the two lines of play. Note that the Encyclopedia does not make this distinction.

Problem 7 (Solutions agree)



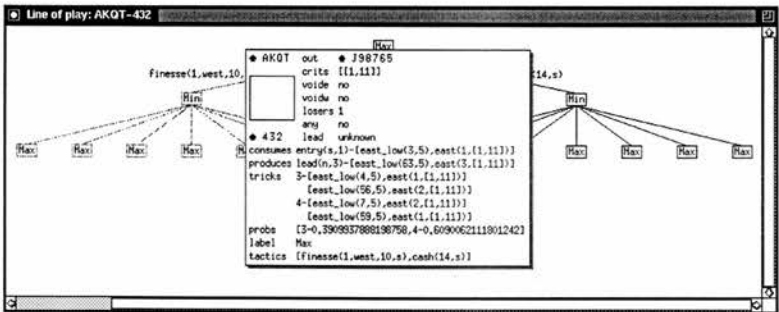
Encyclopedia No.	25
Maximum tricks	4
Critical cards	Jack
Applicable tactics	Type 1 finesse of ten against West Cash Ace
FINESSE prob	0.609
Encyclopedia prob	0.61

FINESSE Explanation:
Cash the Ace. This leads to 4 tricks if East holds the Jack and West holds at least three of the remaining five low cards, or if West holds the Jack and East holds zero, one, three, four, or five of the remaining five low cards.

Encyclopedia Explanation:
Cash the King and Queen; if both follow, play the Ace. This is 2% better than a third round finesse.

Comments

This problem demonstrates that FINESSE can correctly produce lines of play which specify different actions for different possible responses by the defenders (the Ace, King and Queen are all equivalent, so FINESSE’s line of cashing the Ace, King and then the Queen is equivalent to that suggested by the Encyclopedia). Note that earlier versions of FINESSE failed to solve such problems correctly because whilst analysing the game tree bottom-up, at each node they only considered the worlds that were possible *from that point*. This results in any nodes with identical states having the same analysis, irrespective of the play history up to that point.



Problem 8 (Solutions agree)



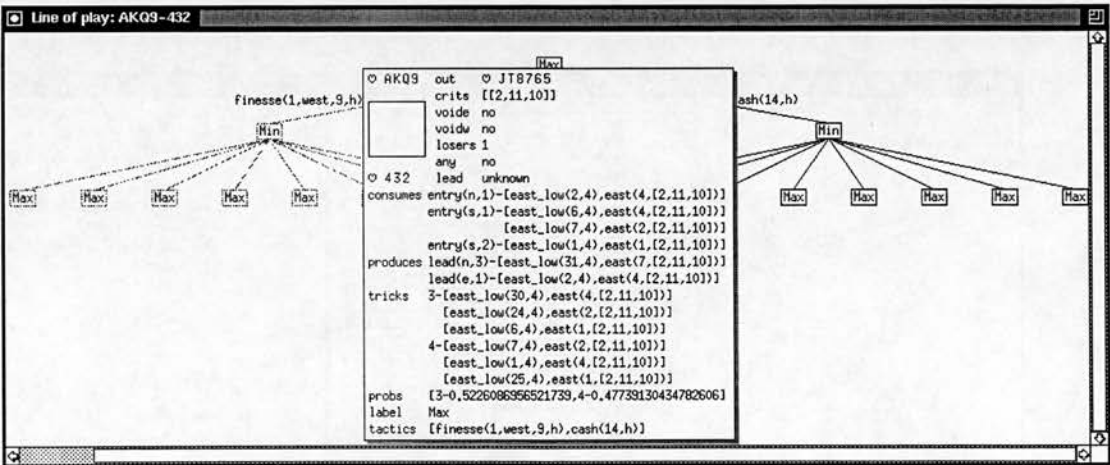
Encyclopedia No.	26
Maximum tricks	4
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of the 9 against West Cash Ace
FINESSE prob	0.477
Encyclopedia prob	0.48

FINESSE Explanation:
Cash the Ace. This leads to 4 tricks if East holds the doubleton Jack, ten, or if the Jack and ten are split evenly between East and West and West holds at least two of the remaining four low cards, or if West holds both the Jack and ten and East holds zero, three, or four of the remaining four low cards.

Encyclopedia Explanation:
Cash the Queen and King; if an honor drops from East, finesse the nine next. This 6% better than cashing the three top honors regardless.

Comments

This problem has already been discussed in Chapter 11. FINESSE’s line of play concurs with that of the Encyclopedia, but on a similar problem, which we present on the following page, FINESSE reveals an error in the Encyclopedia’s analysis.



Problem 9 (Encyclopedia is incorrect)



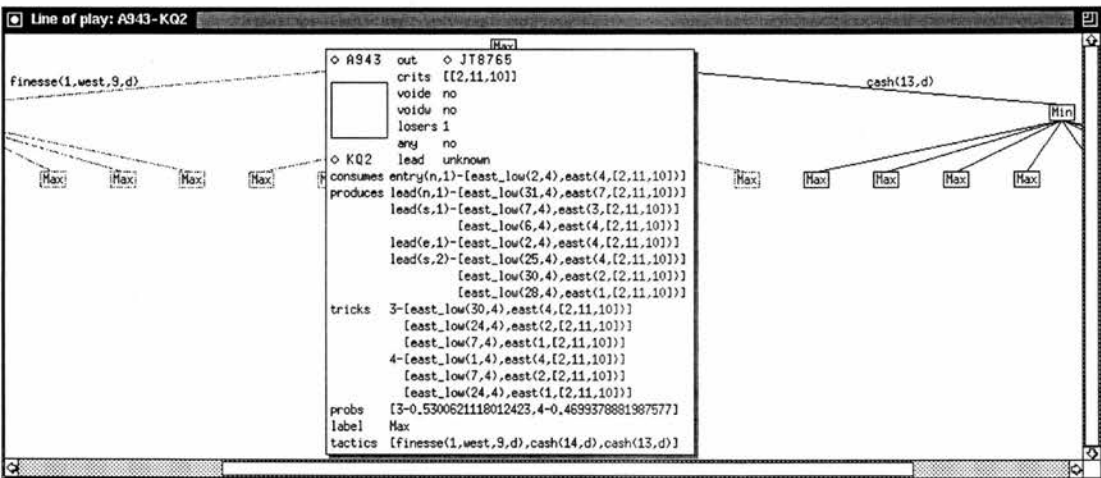
Encyclopedia No.	31
Maximum tricks	4
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of 9 against West Cash Ace Cash King
FINESSE prob	0.470
Encyclopedia prob	0.48

FINESSE Explanation:
Cash the King. This leads to 4 tricks if East holds the doubleton Jack, ten, or if the Jack and ten are split evenly between East and West and West holds at least two of the remaining four low cards, or if West holds both the Jack and ten and East holds at least three of the remaining four low cards.

Encyclopedia Explanation:
Cash the Queen and King; if an honor drops from East, finesse the nine next. This 6% better than cashing the three top honors regardless.

Comments

This problem, which demonstrates an error in the Encyclopedia, has already been discussed in Chapter 11. The Encyclopedia claims that the analysis for this problem is the same as that of problem number 26, which we saw on the previous page. It should be clear that the only difference between FINESSE’s explanations for these problems



is in the last line: the current problem excludes the distribution where 'West holds the Jack and ten and East holds zero... of the remaining four low cards' (*i.e.*, when East is void). It is easily checked that it is indeed impossible to take four tricks in this circumstance since only one finesse against the Jack and ten can be carried out after discovering the break. This discrepancy accounts for the difference in probabilities between FINESSE and the Encyclopedia.

Problem 10 (FINESSE is incorrect)

A K 9 x

N
W E
S

Q 8 x x

Encyclopedia No.	39
Maximum tricks	4
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of 9 against West Type 1 finesse of 8 against East Type 2 finesse of 9 against East Type 2 finesse of 8 against West Cash Ace Cash Queen
FINESSE prob	0.754
Encyclopedia prob	0.79

FINESSE Explanation:
Cash the Queen. This leads to 4 tricks if East holds both the Jack and ten and West holds at least two of the remaining three low cards, or if the Jack and ten are split evenly between East and West and West holds at least one of the remaining three low cards, or if West holds both the Jack and ten and East holds zero, two, or three of the remaining three low cards.

Encyclopedia Explanation:
Cash the Ace. If an honor appears, cash the next top honor from the hand on the left of the J or 10.

Comments

This is the second Encyclopedia problem which is mis-analysed by FINESSE because of non-locality (the first is problem number 5 in this appendix). To understand it, we will examine the line of play recommended by the Encyclopedia, considering in particular what happens when the Ace is cashed and East plays low, with West contributing either the Jack or the Ten. This leads to the following intermediate position.

K 9 x

N
W E
S

Q 8 x

Outstanding:
T x x

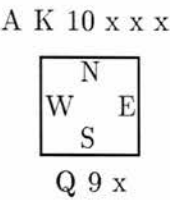
From here, FINESSE opts to cash the Queen, rather than the King, as recommended by the Encyclopedia. The reasons for this error can be seen by examining the table of distributions below. This table shows, for each of the possible distributions of the original outstanding cards from which West can play a critical card and East a low card, whether the cashing of the Queen or King in the intermediate position will lead to a total of four tricks. Notably, cashing the Queen succeeds when East starts with just one low card, because East's void will be revealed when the Queen is played, allowing West to be finessed twice. On the other hand, cashing the King succeeds when West starts with just the singleton Jack or Ten, because it will then be possible to finesse *East* twice.

Distributions	Cash Queen	Cash King
JTxx-x	•	
JTx-xx	•	•
JT-xxx	•	•
Txx-Jx	•	•
Tx-Jxx	•	•
T-Jxxx		•
Jxx-Tx	•	•
Jx-Txx	•	•
J-Txxx		•

FINESSE mistakenly selects the cashing of the Queen because it is more likely to find East with just one low card than to find West with a singleton Jack or Ten. However, this apparent advantage turns out to be illusory, since if West starts with JTxx he can produce a better result by not contributing a critical card when declarer cashes his Ace. Thus, the distributions represented by the first row of the table should in fact be ignored in the intermediate position, because a competent defence would direct the play to a different portion of the search space in this circumstance. Removing them from consideration would result in FINESSE producing the correct line of play.

An important feature of this example is that it demonstrates that non-locality can be produced without the presence of finesse tactics: the only MAX action prior to the intermediate position we discussed above is the cashing of the Ace.

Problem 11 (Solutions agree)



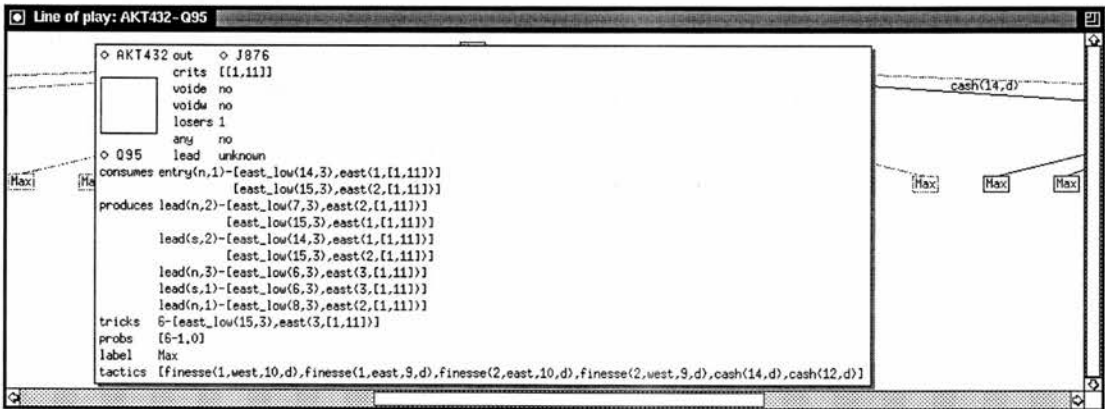
Encyclopedia No.	43
Maximum tricks	6
Critical cards	Jack
Applicable tactics	Type 1 finesse of 10 against West Type 1 finesse of 9 against East Type 2 finesse of Ten against East Type 2 finesse of 9 against West Cash Ace Cash Queen
FINESSE prob	1
Encyclopedia prob	1

FINESSE Explanation:
Cash the Ace. This leads to 6 tricks under any distribution of the outstanding cards.

Encyclopedia Explanation:
Lead the Ace first in case either opponent is void.

Comments

This is an example of a plan with a unity probability of success. However, note that it is important to start by cashing the Ace, and not the Queen, since cashing the Queen will only lead to 5 tricks if West is void. Also, compare this situation to the problem on the next page.



Problem 12 (Solutions agree)

A K 9 x x x

N
W E
S

Q x x

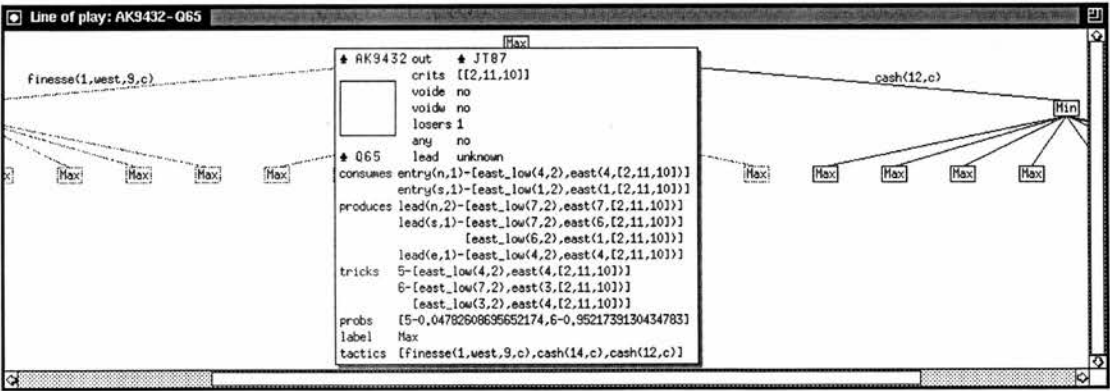
Encyclopedia No.	44
Maximum tricks	6
Critical cards	Jack, Ten
Applicable tactics	Type 1 finesse of 9 against West Cash Ace Cash Queen
FINESSE prob	0.952
Encyclopedia prob	0.95

FINESSE Explanation:
Cash the Queen. This leads to 6 tricks if West holds at least one of the Jack and ten, or if East holds both the Jack and ten and West holds at least one of the remaining two low cards.

Encyclopedia Explanation:
Play queen first, in case East is void

Comments

FINESSE correctly decides to cash the Queen ahead of the Ace. Compare this to the similar situation on the previous page, where the correct play is to cash the Ace ahead of the Queen.



Problem 13 (Solutions agree)



Encyclopedia No.	52
Maximum tricks	4
Critical cards	Queen, Ten
Applicable tactics	Type 1 finesse of 8 against West Type 2 finesse of 8 against East Type 3 finesse of Jack against East Type 4 finesse of Jack against West Cash Ace
FINESSE prob	0.057
Encyclopedia prob	0.06

FINESSE Explanation:
Finesse the Jack (Type 4). This leads to 4 tricks if West holds the Queen, West holds the ten, and East holds at least four of the remaining five low cards, or if East holds the singleton ten.

Encyclopedia Explanation:
Run the jack; if it is covered, finesse the nine next. West must have Q10, Q10x or Qxxxx.

Comments

This is an example of a Type 4 finesse being chosen as the best initial tactic. In this context, this is the equivalent of the Encyclopedia’s direction to ‘run the jack’. The second round finesse which is set up by the Type 4 finesse is a Type 1 finesse of the nine.

Note the correlation between FINESSE’s explanation and that of the Encyclopedia. The distributions where ‘West holds the Queen, West holds the ten, and East holds at least four of the remaining five low cards’ are Q10 and Q10x, and when ‘East holds the singleton ten’, West holds Qxxxx.

Problem 14 (Solutions agree)

Problem 15 (Solutions agree)



Encyclopedia No.	63
Maximum tricks	5
Critical cards	Queen, Ten
Applicable tactics	Type 1 finesse of 9 against West Type 3 finesse of Jack against East Type 4 finesse of Jack against West Cash Ace Duck a round
FINESSE prob	0.071
Encyclopedia prob	0.07

FINESSE Explanation:
Finesse the nine (Type 1). This leads to 5 tricks if West holds the Queen, West holds the ten, and East holds any three of the remaining four low cards.

Encyclopedia Explanation:
Run the jack or lead small to the nine. West must hold Q10x

Comments

In this example, FINESSE again sees a definite distinction between two lines of play that are treated equally by the Encyclopedia. To see why this is so, consider the profiles of the following two lines of play:

- Type 1 finesse of nine [2▷0.080, 3▷0.322, 4▷0.526, 5▷0.071]
- Type 4 finesse of Jack [2▷0.080, 3▷0.371, 4▷0.478, 5▷0.071]

Whilst both have equal chances of making five tricks, FINESSE prefers the finesse of the nine because it has a better chance of producing four tricks in the event that five tricks cannot be taken.

This problem is also a further example of how even FINESSE’s basic explanation system can produce results that are similar to those found in the Encyclopedia.

Problem 16

(FINESSE is incorrect)

A K 9 8

N
W E
S

J x x

Encyclopedia No.	69
Maximum tricks	4
Critical cards	Queen, Ten
Applicable tactics	Type 1 finesse of 8 against West Type 2 finesse of 8 against East Type 3 finesse of Jack against East Type 4 finesse of Jack against West Cash Ace
FINESSE prob	0.240
Encyclopedia prob	0.25

FINESSE Explanation:

Finesse the 8 (Type 1). This leads to 4 tricks if West holds the Queen and West holds the ten.

Encyclopedia Explanation:

Run the jack; if this is covered, finesse the nine.

Comments

In this example, non-locality results in a serious under-estimate of the probability of success for the line of play recommended in the Encyclopedia: 0.107 instead of 0.25. FINESSE therefore selects a different line of play with the slightly inferior probability of 0.240 for producing four tricks (missing only the case where East starts with the singleton Ten).

To understand the circumstances behind FINESSE’s mis-calculation, we need to consider what happens when both defenders contribute a low card on the finesse of the Jack, leading to the following intermediate position:

A K 9

N
W E
S

x x

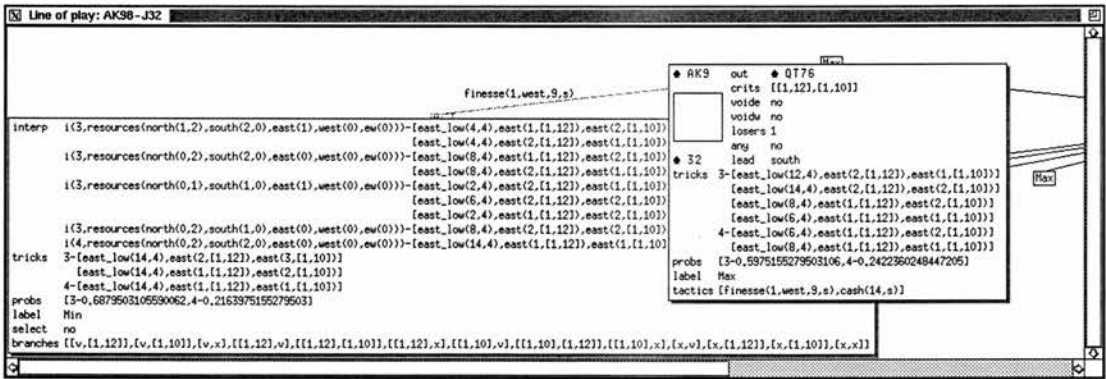
Outstanding:

Q T x x

From here, FINESSE has a choice between finessing the nine (Type 1) and cashing the Ace. The distributions of the original outstanding cards under which these tactics would succeed in producing four tricks are shown in the table on the following page.

Distributions	Finesse of 9	Cash Ace
Qxxx-Tx		•
Qxx-Txx		•
QTxxx-x	•	
QTxx-xx	•	
QTx-xxx	•	•

Based on this, the probabilities of the finesse and the cash are 0.216 and 0.242 as shown by the pop-up windows in the screen capture below (with the right-hand window describing the cash tactic). The cash appears to be the better alternative because with 2 outstanding critical cards it is more likely that they will drop than both be held by West. However, at any node where a defender contributes a critical card rather than a low card, the *finesse* will appear to be the better option. It is this combination of circumstances that causes FINESSE to mis-analyse the problem: considering these situations locally produces a set of choices that work very badly in combination. In particular, when West holds Qxxx or Qxx, he should contribute the Q when declarer begins with the finesse of the Jack, to ensure that the declarer then takes a failing finesse on the second round. Thus, the first two rows of the table above represent distributions under which the defenders should never both play low when declarer starts with he finesse of the Jack. Ignoring these distributions results in the best choice at this node becoming the finesse, which is better than the cash under the splits QTxxx-x and QTxx-xx. The probabilities of these splits is $(4/6 \times 0.0727) + (6/15 \times 0.2422) = 0.145$. This figure is the same as the discrepancy between the Encyclopedia's and FINESSE's assessments of the success of running the Jack.



Problem 17

(FINESSE is incorrect)



Encyclopedia No.	73
Maximum tricks	4
Critical cards	Queen, Ten
Applicable tactics	Type 1 Finesse of 9 against East Type 2 finesse of Jack against West Type 3 finesse of Jack against East Cash the Ace Duck a round
FINESSE prob	0.783 (3 tricks only)
Encyclopedia prob	0.01

FINESSE Explanation:

Cash the Ace. This leads to 3 tricks if East holds the Queen, or if West holds the Queen, West holds the ten, and East holds at least three of the remaining four low cards, or if West holds the Queen, East holds the ten, and West holds zero, two, three, or four of the remaining four low cards.

Encyclopedia Explanation:

Play the Ace, hoping that West has the singleton queen.

Comments

This is the fourth Encyclopedia problem on which non-locality results in FINESSE making an error, and the first (of a total of 12) in which this error results in a line of play which is incapable of taking the maximum possible number of tricks. These examples all have the common form that the correct solution caters for one or maybe two splits of the outstanding cards, such as singleton Kings, doubleton King-Tens, or tripleton Ace-King-Jacks. In the current problem, the situation when four tricks can be taken is when West holds the singleton Queen.

FINESSE misses the optimal line of play despite starting with the correct initial tactic: the cashing of the Ace. When West plays the Queen and East plays low on the first round, however, FINESSE proceeds by cashing the Jack. Locally, this appears to be a good continuation since there is only one critical card remaining with the declarer holding two masters above it. In the table overleaf we show the splits of the original

outstanding cards under which a Type 1 finesse of the 9 (the correct continuation) and the cashing of the Jack lead to four tricks. The probabilities of these distributions are 0.248 and 0.258, so it is the cash that is selected by FINESSE.

Distributions	Finesse of 9	Cash Jack
Qxxx-Tx	•	•
Qxx-Txx	•	•
Qx-Txxx	•	
Q-Txxxx	•	
QTx-xxx		•
QT-xxxx		•

However, consider the situation which is reached when both defenders contribute a low card to the first trick, or when West plays the Ten and East plays low. The best choice from this new state is a Type 3 finesse of the Jack. This (non-local) information has an effect on the above table. Specifically, in any distribution where West holds the Queen but can avoid playing it on the first trick, he should do so (since the finesse of the Jack will then lose on the second round). Thus, the only distribution which should actually be considered in the table is that where West holds the singleton Queen. The finesse of the nine is therefore revealed as the best play at this node, but only after an examination of the tactics selected at other points in the search space.

Problem 18

(Solutions agree)



Encyclopedia No.	153
Maximum tricks	4
Critical cards	King
Applicable tactics	Type 1 finesse of Queen against West Type 2 finesse of Jack against West Type 3 finesse of Jack against East Cash Ace Duck a round
FINESSE prob	0.436
Encyclopedia prob	0.44

FINESSE Explanation:

Finesse the Jack (Type 3). This leads to 4 tricks if West holds the King and East holds any three of the remaining five low cards, or if East holds the King and West holds three or four of the remaining five low cards.

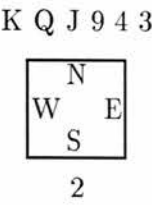
Encyclopedia Explanation:

Lead small to the Jack, hoping that the suit divides 3-3 or that East has Kx.

Comments

This is a further example of the correct initial tactic being a Type 3 finesse. Note that the Encyclopedia’s previous problem (number 152) is identical apart from the replacement of South’s low card by the nine. The optimal line of play for this modified situation is to start with a Type 1 finesse of the nine (described as ‘lead small to the nine’ in the Encyclopedia), and is also correctly identified by FINESSE.

Problem 19 (Solutions agree)



Encyclopedia No.	281
Maximum tricks	5
Critical cards	Ace, Ten
Applicable tactics	Type 1 finesse of 9 against West Type 3 finesse of Jack against West Duck a round
FINESSE prob	0.517
Encyclopedia prob	0.52

FINESSE Explanation:
Finesse the Jack (Type 3). This leads to 5 tricks if either East or West holds the Ace, holds the ten, and the other defender holds at least three of the remaining four low cards, or if either East or West holds the Ace, the other defender holds the ten, and holds two or three of the remaining four low cards.

Encyclopedia Explanation:
Lead to the King; then play the queen and jack.

Comments

Note that in this situation FINESSE’s selection of the finesse of the Jack is equivalent to the Encyclopedia’s ‘lead to the king’, since the declarer holds the sequence King, Queen, and Jack.

The Encyclopedia contains variations on this problem, which FINESSE also solves correctly. For example, moving one of North’s low cards to the South hand results in a decrease in the maximum number of tricks to 4, attainable with a probability of 0.58. Giving North’s Jack to South, on the other hand, makes the optimal line of play (again, for 4 tricks) a lead to the Jack followed by a lead to the King (with a probability of success of 0.54). Also, moving North’s Jack and nine to the South hand, and South’s low card to the North hand requires the 9 to be finessed first, giving a probability of 0.42 of producing 4 tricks. FINESSE solves all these examples correctly.

Problem 20

(Solutions agree)

K Q x x x x x

N

W E

S

—

Encyclopedia No.	436
Maximum tricks	5
Critical cards	Ace
Applicable tactics	Duck a round Sequence: King
FINESSE prob	0.355
Encyclopedia prob	0.36

FINESSE Explanation:

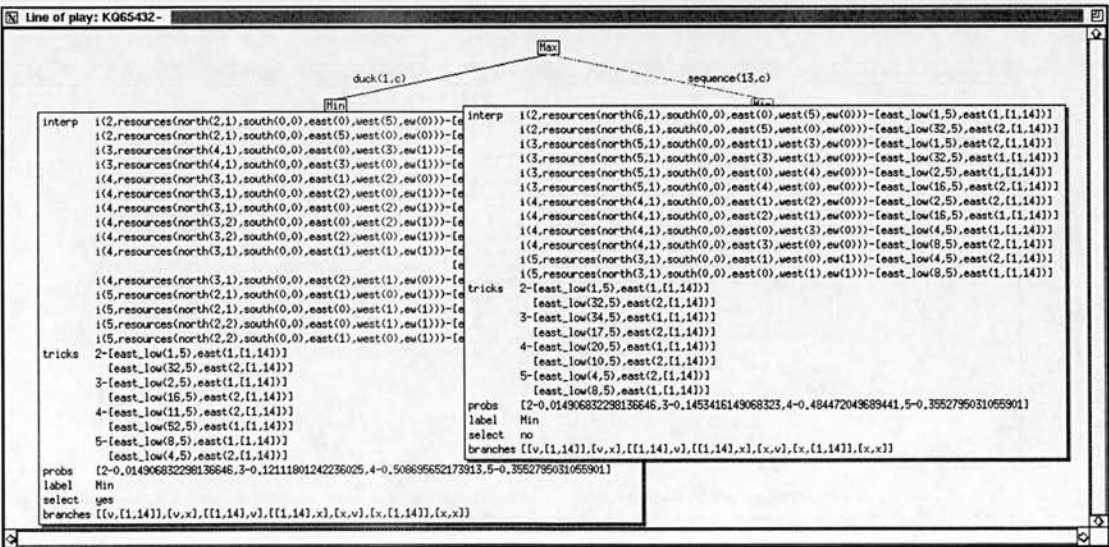
Duck a round. This leads to 5 tricks if either East or West holds the Ace and the other defender holds any three of the remaining five low cards.

Encyclopedia Explanation:

Duck one round, and then play the king (best), or lead the King.

Comments

This example illustrates an instance of the duck tactic being selected as the best initial choice. Note that this is an occasion when the Encyclopedia differentiates between two lines of play even though they have the same chance of producing 5 tricks. The screen capture below explains this: the profile of the sequence tactic (in the right-hand pop-up window) has a lower chance of making four tricks, if five tricks cannot be taken.



Problem 21 (Solutions agree)



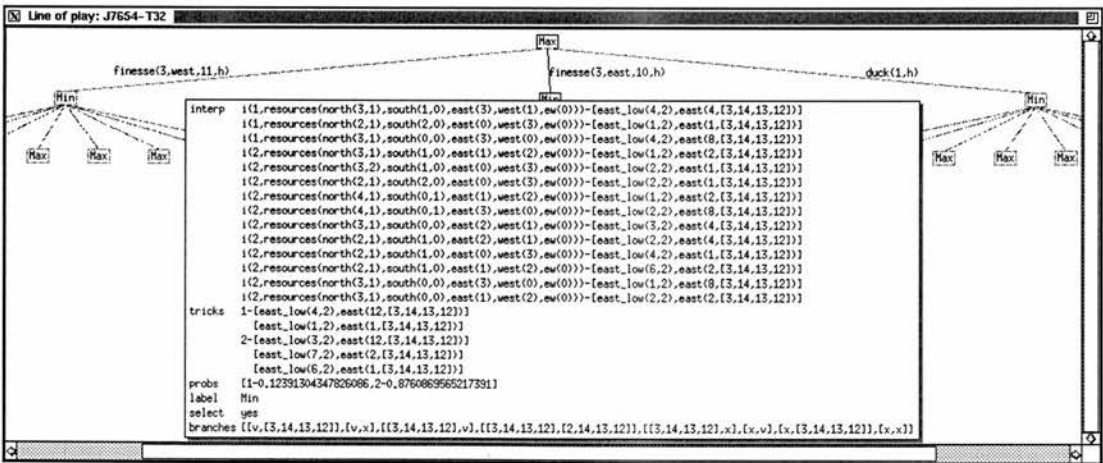
Encyclopedia No.	654
Maximum tricks	2
Critical cards	Ace, King, Queen
Applicable tactics	Type 3 finesse of Jack against West Type 3 finesse of 10 against East Duck a round
FINESSE prob	0.876
Encyclopedia prob	0.88

FINESSE Explanation:
Finesse the Ten (Type 3). This leads to 2 tricks if East holds at least two of the Ace, King, and Queen and West holds at least one of the remaining two low cards, or if West holds any two of the Ace, King, and Queen, or if West holds the Ace, King, and Queen and East holds at least one of the remaining two low cards.

Encyclopedia Explanation:
Lead small to the ten. If it loses to West, lead small to the Jack.

Comments

In this example, the top three cards are held by the defence. However, it is still possible to produce two tricks, and FINESSE finds the optimal line of play for doing so.



Problem 22

(Solutions agree)



Encyclopedia No.	655
Maximum tricks	1
Critical cards	Ace, King, Queen
Applicable tactics	Type 3 finesse of Ten against West Duck a round
FINESSE prob	0.839
Encyclopedia prob	0.84

FINESSE Explanation:

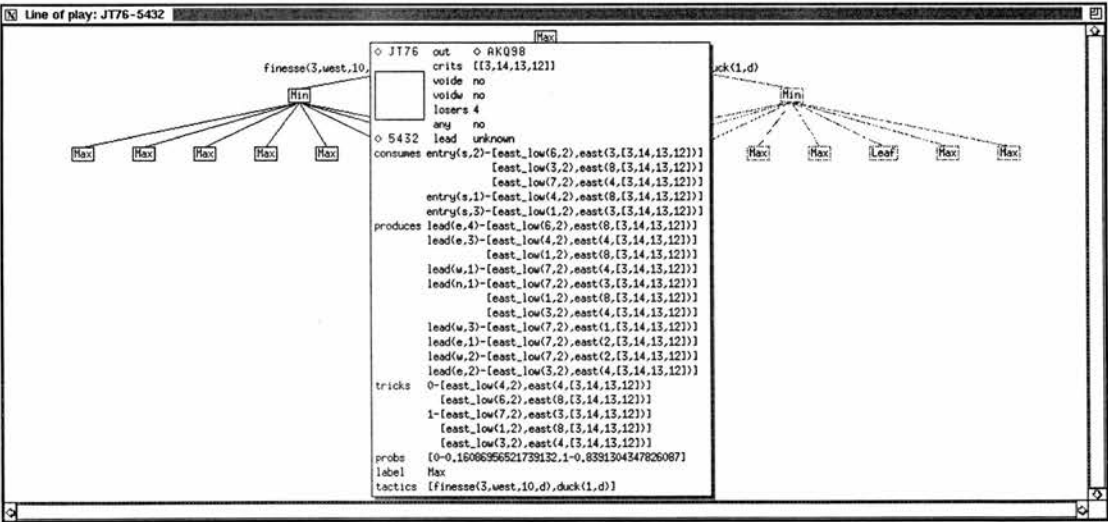
Finesse the Ten (Type 3). This leads to 1 trick if West holds at least two of the Ace, King, and Queen, or if East holds just the Ace, King, and Queen, or if East holds any two of the Ace, King, and Queen and West holds at least one of the remaining two low cards.

Encyclopedia Explanation:

Lead small to the jack and then small to the ten.

Comments

Declarer’s longest hand now only has four cards, so with three outstanding masters only one trick can be taken.



Problem 23 (Solutions agree)



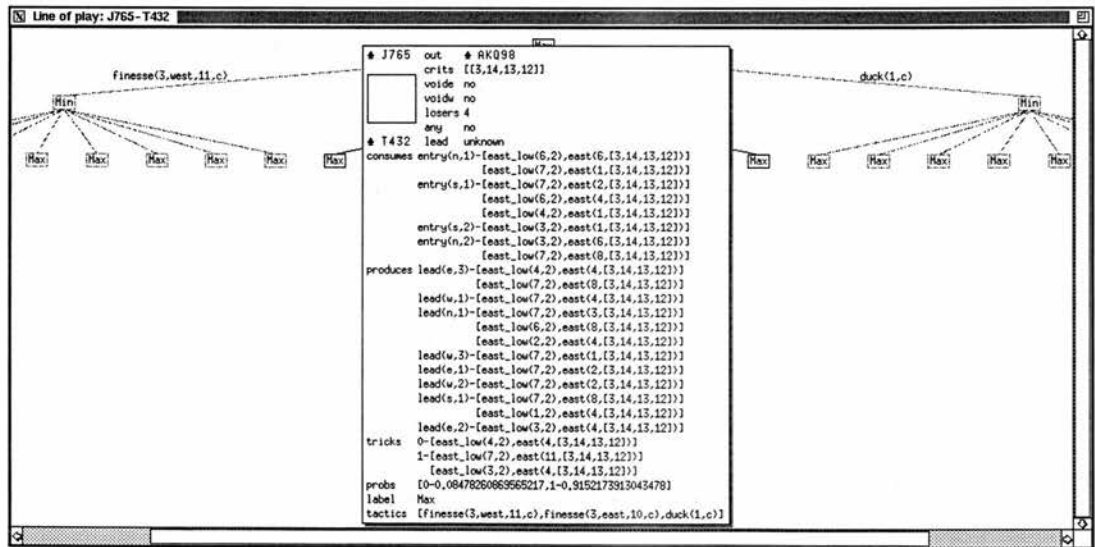
Encyclopedia No.	656
Maximum tricks	1
Critical cards	Ace, King, Queen
Applicable tactics	Type 3 finesse of Jack against West Type 3 finesse of 10 against East Duck a round
FINESSE prob	0.915
Encyclopedia prob	0.92

FINESSE Explanation:
Finesse the Ten (Type 3). This leads to 1 trick if West holds zero, two, or three of the Ace, King, and Queen, or if East holds any two of the Ace, King, and Queen and West holds at least one of the remaining two low cards.

Encyclopedia Explanation:
Lead small to the jack (or ten). If it loses, lead small to the other honor.

Comments

With the Jack and ten in different hands, FINESSE correctly finds that it has a higher probability of winning one trick than in the previous problem. The differences between the analysis of this and the previous two problems illustrate the power of a qualitative reasoning system to distinguish the actual distributions under which any line of play will succeed.



Appendix D

User Manual

When all else fails, read the manual.

— ANON

This appendix summarises the user-level predicates provided by the actual implementation of FINESSE. To begin with, §D.1 gives a basic overview of the design and §D.2 gives instructions for starting the system. §D.3, §D.4, and §D.5 then describe the predicates for setting up games, for altering and displaying the state of a game, and for interfacing to the planning algorithms. Since the distribution version of FINESSE also includes an automated bidding system written by David Asher, a description of the interface predicates to this package is given in §D.6. Finally, some notes on the actual implementation and on the form of the saved game files are included in §D.7 and §D.8.

D.1 Implementation Overview

FINESSE modifies the top-level environment of Prolog to support the basic manipulation of a data structure that represents a Bridge game. The implementation of this environment can be seen as an extension of Prolog by an abstract data type. Simple predicates are provided to change the state of this data structure, and to save or load the current state to or from a file. FINESSE's planning predicates take their information on the current state of the game from the value of this data structure.

The design of this interface was based on the OYSTER [Bundy *et al* 90b] theorem-proving environment developed at Edinburgh University. However, whereas in OYSTER

the theorem-proving domain naturally leads to the generation of proof *trees*, the record of a game of Bridge is more like a *sequence* of states. Every time a card is played, a new situation is created, but the previous state is also retained. Any state from the sequence that records the play of a game is recoverable from the entries stored in the database. The state which is viewed as the ‘current’ state within the sequence is determined by the value of the *focus*, which can be modified by the user. Backtracking to previous states of play will of course not be possible for a Bridge-playing program, but human users may find it useful to revise the play of a game, or to inspect the decisions made by the planners or bidding system.

D.2 Initialising the System

Once the code has been properly installed (see Appendix E), the system can be started by changing to the top-level FINESSE directory, running SICStus Prolog (a version later than 2.1.5, if the graphical interface is to be used), and consulting the file `init.pl`.

FINESSE has a default initialisation file, `.xfinrc`, which resides in the top-level FINESSE directory. Individual users wishing to customise the program, however, can create a copy of this file and place it in their home directories. When `init.pl` is consulted, the `.xfinrc` file in the user’s home directory will be read instead of the generic original. Currently, the main use of this feature is to add extra directories to the search path for saved games. This enables users to create libraries of saved games in their own file space and still access the saved games that are part of the original distribution.

D.3 Setting up Play Situations

The following predicates allow users to save or load games to or from files. Also included are predicates allowing the user to input game states by typing each players’ cards, and for generating random game states.

-
- `save_game(+FileName) / load_game(+FileName)`
-

Save/load the sequence of stored sates to/from the file with local name `FileName`.

- `lib_save(+FileName) / lib_load(+Filename)`

Perform the same saving or loading operations, but this time using the directories indicated by the setting of the `saved_game_file_path` environment variable in the initialisation file (`.xfnrc`).

- `ls`

List the contents of the current library directories.

- `enter_hands`

Starts a simple text input loop to enable the user to type in a specific deal of cards.

- `start_rand`

Generates a random deal, with West as the dealer.

- `start_rand(+Dealer)`

Generates a random deal with `Dealer` as the dealer (may be `n`, `e`, `s`, or `w`).

- `randomise`

Allows the user to select a new random number seed.

D.4 Basic Interface

The top-level environment supplies the following predicates:

- `help`

A basic, text-based help system summarising the main system functions and commands.

- `first`

Moves the focus to the start of the game, before any bidding or card play.

- **last**

Moves the focus up to the last state in the stored sequence.

- **middle**

Moves the focus to the state where the first card play is made (*i.e.*, between the bidding and the play phases of the game). Fails if no contract has been specified.

- **forward**

Moves the current focus one position forward. Fails if this move would result in going past the end of the sequence of stored states.

- **forward(+N)**

Moves the focus forward through N states, or up to the end of the sequence of states, whichever occurs first.

- **back**

Moves the current focus one position backwards. Fails if the current focus is at the start of the game.

- **back(+N)**

Moves the focus backwards through N states, or up to the beginning of the sequence of states, whichever occurs first.

- **play(+Card, +Suit)**

Behaves like a deterministic Prolog predicate with the side-effect of updating the stored sequence of game states. Its effects include removing the specified card from the current player's hand (the predicate fails if this cannot be done), keeping track of the trick count, and moving the focus forward to the next state. These side-effects are not undone under backtracking.

- `set_contract(+Bid, +Player)`

Deletes all the previous play and bidding history, moves the focus to the start of play, and sets the declarer to be `Player` (either `n`, `e`, `s` or `w`) and the contract to be `Bid` — a Prolog term of the form `Number - Suit`, where `Number` is an integer between 1 and 7, and `Suit` is either `s`, `h`, `d`, `c` or `nt`. (Notice that doubled and redoubled contracts cannot currently be set.)

- `display`

Produces a text-based display of the *current* game state (as determined by the current focus).

- `display_trick(+N)`

Produces a display of all the cards played in the specified trick. If the current focus falls within the trick it is indicated by a `^` beneath the current player's symbol. Fails if the start of trick `N` lies beyond the limit of the sequence of stored states.

- `display_bidding`

Produces a display of the entire bidding history. If the current focus lies in the bidding phase, it is indicated by a `^` beneath the current player's symbol.

D.5 Interface Predicates to Planners

The following predicates interface to the planning system. These predicates all determine the 'current' state of the game by examining the value of the focus.

- `ic(+Suit)`

Interface to the pre-planner: writes the list of critical cards in `Suit` to stdout.

- `ics`

Writes the list of critical cards in all four suits to stdout.

• **app(+Suit)**

Query the method set. All the tactics applicable to the current state in the given **Suit** are written to **stdout**.

• **apps**

The applicable tactics in each suit of the current game state are written to **stdout**.

• **xplan(+Suit, +Type)**

Interface to the planner: causes a plan for **Suit** (either **s**, **h**, **d** or **c**) to be produced and displayed as a tree in a gm window. The algorithm used for planning is determined by **Type**, which should be one of the following atoms:

old	Old version of planner
trace	Trace while planning
history	Keep a history for efficiency
interpret	Keep a history and interpret results
top	Keep a history, but only interpret cash tactics

• **enc(+Num)**

Provided the library directory containing problems from the Bridge Encyclopedia is present (this is the default), this predicate loads the file containing problem number **Num**, forms a plan, interprets the results and then displays the output in a graphical window.

• **play_game**

Interface to the interleaver: causes each suit to be planned and interpreted, and the resulting plans to be interleaved. The solution constructor trees resulting from each interleaving stage are displayed in windows, provided the gm process is running.

• **play_style(+Style)**

Sets the playing style to **Style**. **FINESSE** currently has four styles which choose

between global plans on the basis of the following criteria:

<code>expected</code>	Highest expected value
<code>conservative</code>	Smallest chance of making lowest number of tricks
<code>aggressive</code>	Highest chance of winning the largest number of tricks
<code>make_contract</code>	Highest chance of making at least enough tricks for contract

• `play_style`

Reports the current playing style setting to stdout.

• `non-locality`

Toggles FINESSE's non-locality heuristic on or off.

• `xfin`

Start a top-level graphical interface which provides buttons for invoking each of the five types of planning operation, planning of the game state, loading games, and a help window.

D.6 Interface Predicates to Bidding System

A prototype bidding system was added to the FINESSE environment by David Asher [Asher 93]. Asher's system uses a set of bidding rules based on the ACOL bidding system, but is designed primarily for auctions with no competition. The following predicates allow bids to be made and the reasons behind bidding decisions to be examined.

• `context`

Displays the *context* of the next bid to be made by the current player (the rules in the bidding database are divided into groups according to the context for which they are designed).

• `rules`

Displays the bidding rules applicable for the next bid of the current player.

- **bid**

Makes an automatic bid for the current player (East and West are always assumed to pass).

- **bid(+Bid)**

Makes the specified **Bid** for the current player, where **Bid** may be **pass**, **dbl**, **redbl**, or a term of the form **N-Suit**, where **N** is a number from 1 to 7, and **Suit** is either **s**, **h**, **d**, **c** or **nt**.

- **all**

Bid for all players, from the current game position until a contract has been reached.

- **infer**

Display inferences about the last bid.

- **inferences**

Display all inferences that can be made about all previous bids.

D.7 Notes on Implementation

The properties which define the state of a Bridge game are stored in Prolog's database, from where they are accessed or updated by the infix operators **==>** and **<==** respectively. The effects of these operators are:

- **K<==V** stores value **V** under the key **K**. The entry is further indexed by the value of the current focus. It is this indexing that enables any state passed through during the play of a game to be recalled. The keys **cpos** and **limit** and **last_bid** are exempt from this indexing, essentially being used as global variables.
- **K==>V** binds **V** to the 'most recent' entry stored under the key **K**. The phrase 'most recent' indicates that any entry stored under an index which represents a later game state than the current focus is not considered.

Figure D.1 gives a complete list of the keys currently utilised. However, should extra functionality be desired (for example, the storing of inferences drawn from the bidding) further keys can be added with minimal disruption.

Key	Information	Updated at
dealer	dealer	start only
lead	player on lead	each trick
south_cards	South's hand	each play by South
north_cards	North's hand	each play by North
out_cards	combined holding of East and West	each play by E/W
north_card	North's card on the current trick	each play by North
east_card	East's card on the current trick	each play by East
south_card	South's card on the current trick	each play by South
west_card	West's card on the current trick	each play by West
tricks_for	number of tricks won by declarer	end of each trick
strongest_player	strongest player in current trick	each play
strongest_card	strongest card in current trick	each play
player	next player to bid or play	each action
north_bid	North's bid on current round	each bid by North
east_bid	East's bid on current round	each bid by East
south_bid	South's bid on current round	each bid by South
west_bid	West's bid on current round	each bid by West
consecutive_passes	number of consecutive passes	each bid
strongest_bid	strongest bid so far	each bid
bid_status	is bid doubled or redoubled?	each bid
cpo	current focus	each action
limit	end of sequence of States	each action
last_bid	end of sequence of bids	each bid

Figure D.1: Database keys used to store state of game

Six further database keys, shown in Figure D.2 on the next page, are used by FINESSE to record the current state of the user's interaction with the system. These, however, are not considered part of the basic data structure (and are thus not saved or reset when new games are loaded or saved).

D.8 Saved Games

A template file for entering saved games is listed at the end of this section. The minimum information that must be specified is the dealer and the cards held by North and

Key	Purpose
gm_interface_in_use	Yes/no flag for graphics manager (gm) interface
gm_view_window_key	Dummy number for generating unique gm window identifiers
trace	Records whether planning trace is switched on
creep	Records whether creep is switched on
play_style	Criterion used to select the best global plan
non_locality	Records whether non-locality heuristic is being used

Figure D.2: Additional database keys used by FINESSE

South. If the outstanding cards are not specified they will be calculated automatically, but end-game situations can also be entered by specifying the outstanding cards as well as those of North and South. Another way to create end-game situations is to enter the initial deal for the hand (either using the template file or the `enter_hands/0` predicate) and then specify the cards played on each known trick of the game using the `play/2` predicate at the interface top-level.

Note that the operator used to separate key-value pairs in saved game files is actually `<--` and not `<==`. This is because the `<==` operator indexes data by the value of the current focus. In saved game files generated by the system, however, each item of data will already have an index, which should not be changed when the game is loaded. The `<--` operator therefore stores data whilst preserving its original index, or if there is no index present (such as in the template file), by assuming that the data should be indexed with the start position of the game.

```
%
%                               File:      template.pl
%                               Author:    Ian Frank
%                               Date:      May 21st 1991
%                               Purpose:   template for entering games of bridge.
%                               Modified:  25 June 1993 to meet spec of new automaton

dealer <-- west.    % Must set a dealer (west/east/south/north)

/*****
* The cards for each player are stored as four terms of the form:      *
*      suit(Length, SuitName, Cards)                                   *
* Aces are represented as 14, Kings as 13, etc....                      *
* EXAMPLE:                                                              *
*      suit(4, s, [13 ,11, 9, 3]) represents a holding                 *
*      of the King, Jack 9 and 3 of spades.                             *
*****/
north_cards <-- [ suit(0, s,[]),
```

```

        suit(0, h,[]),
        suit(0, d,[]),
        suit(0, c,[]) ].
south_cards <-- [ suit(0, s,[]),
        suit(0, h,[]),
        suit(0, d,[]),
        suit(0, c,[]) ].

/*****
* Specifying 'out_cards' is optional (unless you want to specify a game *
* which doesn't start from the first trick - e.g, an end-game position) *
*****/
%out_cards <-- [ suit(, s,[]),
%               suit(, h,[]),
%               suit(, d,[]),
%               suit(, c,[]) ].

end.
```


Appendix E

Code

‘Well, though it seems
Beyond our dreams,’
Said Littell to Scott,
‘We’ve really got
To the very end.’

— THOMAS HARDY
Littell and Scott

The original plan was to end by listing the code for the FINESSE system in this appendix. Then we found out that, even excluding the portions for the graphical interfaces, this would take over 130 pages. Since the world state we find ourselves in doesn’t have the resources to cope with this, we fall back on the alternative plan of making the code available for anonymous download. Instructions for retrieving the appropriate files are given below.

E.1 Ftp Instructions

The code is available by anonymous ftp from the ftp site of the Mathematical Reasoning Group at Edinburgh University (`dream.dai.ed.ac.uk` [192.41.111.169]) in the directory `pub/misc`.

E.1.1 Step by Step Instructions

To retrieve a copy of the code from the ftp server please follow this example.

```
% ftp dream.dai.ed.ac.uk                (or ftp 192.41.111.169)

220 achtriochtan FTP server (SunOS 4.1) ready.
Name : anonymous
Password:                               (please enter your e-mail address)
                                           (it is not seen on the screen   )

ftp> cd pub/misc
ftp> binary
ftp> get Finesse1.0.0.tar.Z      (or latest numbered version )
ftp> quit

% uncompress Finesse1.0.0.tar.Z
```

You now have a tar file `Finesse1.0.0.tar` which can be extracted into the code for the system. If you do not have 'uncompress' or 'tar' please contact your system administrator.

E.2 Getting Started

The tar file itself comes with instructions on how to use the system. For a quick start, change to the directory containing the code, edit the path names in the file `.xfinc` to reflect your system, run SICStus Prolog (a version later than 2.1.5 will be necessary if you wish to make use of the graphical interfaces), and consult the file `init.pl`.

Glossary

‘And now we need as it were a tompion to protect the contents of this flask from invading bacteria. I presume you know what a tompion is, Cornelius?’

‘I can’t say I do, sir’, I said.

‘Can anyone give me a definition of that common English noun?’ A. R. Woresley said.

Nobody could ...

‘Oh, come on, sir’, someone said. ‘Tell us what it means.’

‘A tompion’, A. R. Woresley said, ‘is a small pellet made out of mud and saliva which a bear inserts into its anus before hibernating for the winter, to stop the ants getting in.’

— ROALD DAHL
My Uncle Oswald

Agent

Any entity (*e.g.*, a robot, a person, or an adversary) that is situated in a domain that it reacts to and acts upon.

Action

An event that happens in a domain, usually carried out by one of the **agents** in that domain, and probably resulting in a change in the state of the domain.

Behavioural Strategy

Whereas a **pure strategy** specifies *exactly one* action for each situation that could possibly arise during the course of a game, a behavioural strategy can specify *multiple* possible actions for each situation. In order to choose between these multiple actions, a behavioural strategy also specifies the probability that each action will be selected in each possible situation.

Best Defence Model

The best defence model of a two-player game restricts MAX to a pure **strategy** which he must choose before MIN. The outcome of every move in the game (including chance moves) is also known to MIN as soon as it happens. The best defence model of Bridge is often (implicitly) assumed in Bridge books when describing the best **lines of play** in given situations.

Card Combination

The cards held in one suit by one of the sides in a Bridge game. Bridge books sometimes discuss how particular card combinations should be tackled to produce the best results (since ignoring the situation in the other suits makes the task much simpler).

Cash

The playing of a **master** in some suit. In situations where there is no **trump suit**, the master will always win the trick.

Cracked

A game is cracked when a computer program is capable of always achieving the best possible theoretical results in the game, whatever opposition it faces. Compare with **solved**.

Critical Card

In a single suit, any outstanding card whose position may affect the chances of winning tricks in the suit.

Discard

The play of a card from a different suit to the one that started the trick. This play always loses (assuming the card played is not a **trump**), so discards tend to be low cards.

Duck

Contributing a low card to a trick when your hand contains other cards that may be capable of winning it.

Duplicate Bridge

A form of Bridge where many players or teams have their performance compared on the same sets of cards. Unlike in **rubber Bridge**, making overtricks wherever possible is usually critical in order to do well.

Eight Puzzle

A toy puzzle featuring eight sliding tiles in a little 3×3 grid. Seems to appear in everything you've ever read about AI. Including this thesis, now.

Extended Operator Representation

An **operator** representation of **actions** that are more complex than the simple blocks world type actions. Actions which call for such representations include those with context-dependent effects (*i.e.*, their effects cannot be represented by some universal lists of domain properties which they alter), or those whose outcomes depend on unknown or uncertain information.

Finesse

An attempt to develop extra tricks in a suit by winning with cards that are not **masters**. Finesses rely on the elementary principle of card play that the best results can be obtained by forcing an opponent to play ahead of you.

Information Set

The set of possible game situations which are consistent with the play so far. In many games, (for example, noughts and crosses or chess) any player's information set will contain only one member: the current state of the game. However, in games where the outcome of some moves can be hidden, the information set of a player may contain more than member: one for each of the sequences of moves which appears possible to the player.

Inter-suit Tactic

A Bridge **tactic** that describes useful **actions** involving cards from more than one suit.

Interpreter Algorithm

Algorithm used by FINESSE to identify promising **lines of play** in the trees of **tactics** produced by the single-suit planner.

Legal Play Sequence (lps)

Any sequence of moves which forms a complete play of a game, without breaking any of the game's rules.

Line of Play

A modified version of a **strategy**. Rather than specifying a *single* choice for each possible position that may arise, multiple choices can be allowed, providing they lead to the same payoff under every possible **world**.

Losers

Either in a single suit or a complete hand, the difference between the number of cards held and the number of tricks that can be won by playing **masters**.

Master

Any card which is higher than all the opponents' cards in the same suit.

Method

A heuristic specification of the pre-conditions of a **tactic**. Such specifications are typically used by a planning algorithm to decide when the corresponding tactics can be applied.

Mixed Strategy

A player using a mixed strategy for some game must first specify some number of **pure strategies** for the game, and also a probability vector describing the chance that he will use each of these pure strategies when the game is actually played. This type of probabilistic strategy prevents the opponents from using their knowledge of a player's strategy to improve their own chances.

Non-locality

Phenomenon caused by differing information between players in games with incomplete information. For such games, the best choice of move at any node in the tree of possibilities is not simply a function of the subtree of that node, but also of all the other nodes in the tree.

Operator

Any internal representation of a domain **action**. May be either *simple* (representing single domain actions) or *compound* (representing multiple actions).

Payoff

The value that a player attaches to the play of a game.

Proof-planning

A technique for finding proofs for mathematical theorems. The possible operators available at any stage of planning are restricted to a set of **tactics**, whose preconditions are specified as **methods**.

Pure Strategy

A pure strategy determines a course of action for a single player in a game by specifying *exactly one* action for each situation that could possibly arise during the course of the game.

Rubber Bridge

A form of Bridge where winning largely depends on scoring enough points to win two 'games' of 100 points. Points can only be earned towards these games by making contracts. Other bonuses are available, but typically the main aim during card play is to maximise the chances of succeeding in whatever contract was bid.

Ruffing

The playing of a card from the **trump suit** on a trick which was started by a card from a different suit. The player playing the highest trump card on the trick is the winner.

Sequence Tactic

A play carried out in Bridge when the declarer has no **masters** in a suit but his highest sequence of cards is longer than the highest outstanding sequence of cards in the suit.

Solved

A game is solved when the best **strategy** can be explained in human terms. Compare with **cracked**.

Squeeze

A play in Bridge which involves forcing one defender to **discard** one of the two cards he holds that are stopping the declarer winning a trick.

Strategy

A course of action for a single player in a game, specifying the move to be made by that player in any situation that could possibly arise during the course of the game. Possible types of strategy include **pure strategies** and **mixed strategies**.

Tactic

A specific **operator** (simple or compound) that has been identified as being either useful or sensible in a particular situation.

Top Tricks

The tricks which can be won by simply **cashing** cards which are **masters**.

Truth Criterion

An algorithm for determining whether any given proposition is true at some point of a plan.

Trump Suit

A suit which is stronger than all the others. When any trumps are played on a trick, the highest will be the winner, irrespective of the values of the other cards, or which card started the trick.

Void

A player with no cards in a particular suit is said to be void in that suit.

Vulnerability

In **rubber Bridge** one side is vulnerable if they have won one game. A side which is vulnerable suffers greater penalties if they fall short of a contract which they bid.

Worlds

In a game with chance moves, each possible (pure) **strategy** for the chance player determines a possible world in which the play may take place.

Bibliography

- [ACBL 94] ACBL. *The Official Encyclopedia of Bridge*. American Contract Bridge League, Inc., 2990 Airways Boulevard, Memphis, Tennessee 38116-3875, fifth edition, 1994.
- [Allen 89] J.D. Allen. A note on the computer solution of connect-four. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 134–135. Ellis Horwood, 1989.
- [Allis 92] L.V. Allis. Qubic solved again. In J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*, pages 192–204. Ellis Horwood, 1992.
- [Allis et al 91] L.V. Allis, H.J. van den Herik, and I.S. Herschberg. Which games will survive? In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 232–243. Ellis Horwood, 1991.
- [Ambros-Ingerson & Steel 88] J Ambros-Ingerson and S Steel. Integrating planning, execution, and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83–88, St Paul, MN, 1988. AAAI. Also in Allen, J., Hendler, J. and Tate, A., eds., *Readings in Planning*, Morgan Kaufmann, 1990.
- [Asher 93] David Asher. Convent — a bridge bidding system. Unpublished M.Sc. thesis, DAI, 1993.
- [Barret & Weld 93] A. Barret and D.S. Weld. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1993.
- [Barrett et al 91] A Barrett, S Soderland, and D S Weld. Effect of step-order representations on planning. Technical Report 91-05-06, Department of Computer

- Science and Engineering, University of Washington, Seattle, 1991.
- [Berlekamp 63] E.R. Berlekamp. Program for double-dummy bridge problems — a new strategy for mechanical game playing. *Journal of the ACM*, 10(4):357–364, 1963. Also in D. Levy, editor, *Computer Games II*. Springer Verlag, New York, 1988.
- [Berlin 85] Daniel L.S. Berlin. Span: Integrating problem solving tactics. In *Proceedings of IJCAI-85*, pages 1047–1051, Los Angeles, California, 1985.
- [Berliner 80] H. Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14:205–220, 1980.
- [Boddy & Dean 89] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, Detroit, MI, 1989. AAAI.
- [Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.
- [Bundy 91] A Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.
- [Bundy 92] A. Bundy. Incidence calculus. *Encyclopedia of Artificial Intelligence*, pages 663–668, 1992. Also available from Edinburgh as DAI Research Paper No. 497.
- [Bundy et al 90a] A. Bundy, A. Smaill, and J. Hesketh. Turning eureka steps into calculations in automatic program synthesis. In S.L.H. Clarke, editor, *Proceedings of UK IT 90*, pages 221–6, 1990. Also available from Edinburgh as DAI Research Paper 448.
- [Bundy et al 90b] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on*

- Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.
- [Bundy *et al* 91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.
- [Carley 62] Gay Carley. A program to play contract bridge. Unpublished M.Sc. thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1962.
- [Chapman 87] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987. Also in Allen, J., Hendler, J. and Tate, A., eds., *Readings in Planning*, Morgan Kaufmann, 1990.
- [Collins & Pryor 95] Gregg Collins and Louise Pryor. Planning under uncertainty: Some key issues. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1567–1573, Montreal, Canada, 1995. IJCAI.
- [Counsell 94] Gail Counsell. It's not like playing with Omar. *The Independent*, October 31 1994.
- [Currie & Tate 85] K. Currie and A. Tate. O-plan: Control in the open planning architecture. In *Proceedings of the BCS Expert Systems Conference '95*. Cambridge University Press, 1985.
- [Davey 78] Anthony Davey. *Discourse production : a computer model of some aspects of a speaker*. Edinburgh University Press, 1978.
- [Dean & Boddy 88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, St Paul, MN, 1988. AAAI.
- [Draper *et al* 94a] Denise Draper, Steve Hanks, and Daniel Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 178–186, Seattle, WA, 1994. Morgan Kaufmann.

- [Draper *et al* 94b] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 31–36, Chicago, IL, 1994. AAAI Press.
- [Dreyfus & Law 77] S.E. Dreyfus and A.M. Law. *The Art and Theory of Dynamic Programming*. Academic Press Inc., 1977.
- [Drummond & Tate 87] M. Drummond and A. Tate. AI planning. In H. Adeli, editor, *Knowledge Engineering, Vol. 1, Fundamentals*. McGraw-Hill, 1987.
- [Drummond 89] Mark Drummond. Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–113, Toronto, 1989. Morgan Kaufmann.
- [Erbach 92] David W. Erbach. Computers and go. In Richard Bozulich, editor, *The Go Player's Almanac*. The Ishii Press, 1992. Chapter 11.
- [Ernst & Newell 69] G. Ernst and A. Newell. *GPS: a case study in generality and problem solving*. Academic Press, New York, 1969.
- [Erskine 92] Christina Erskine. Omar Sharif's Bridge. *PC Review*, pages 58–59, July 1992.
- [Etzioni *et al* 92] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 115–125, Boston, MA, 1992. Morgan Kaufmann.
- [Fikes *et al* 72] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972. Also in Allen, J., Hendler, J. and Tate, A., eds., *Readings in Planning*, Morgan Kaufmann, 1990.
- [Forrester & Dimbleby 95] Tony Forrester and Henry Dimbleby. Games: When winning is no big deal. *Daily Telegraph*, Jan 25 1995.
- [Frank 89] A. Frank. Brute force search in games of imperfect information. In D.N.L. Levy and D.F. Beal, editors,

- Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 204–209. Ellis Horwood, 1989.
- [Frank 91] I. Frank. Finesse: An adaption of proof-planning techniques to declarer play in the game of Bridge. Unpublished M.Sc. thesis, DAI, 1991.
- [Frank *et al* 92] I. Frank, D. Basin, and A. Bundy. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, pages 72–76, Vienna, Austria, 1992. Longer Version available from Edinburgh as DAI Research Paper No. 575.
- [Fudenberg & Tirole 95] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1995.
- [Gambäck *et al* 91] Bjorn Gambäck, Manny Rayner, and Barney Pell. An architecture for a sophisticated mechanical bridge player. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 87–107. Ellis Horwood, 1991. Later version also available as Cambridge University Computer Laboratory Tech Report 299, May 1993.
- [Geake 92] Elizabeth Geake. Playing to win. *New Scientist*, pages 24–25, 19 September 1992.
- [Ginsberg 95] M. Ginsberg. How computers will play bridge. *The Bridge World*, 1995. To appear. Also available for anonymous ftp from dt.cirl.uoregon.edu as the file /papers/bridge.ps.
- [Goldman & Boddy 94a] Robert P Goldman and Mark S Boddy. Conditional linear planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 80–85, Chicago, IL, 1994. AAAI Press.
- [Goldman & Boddy 94b] Robert P Goldman and Mark S Boddy. Representing uncertainty in simple planners. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 238–245, Bonn, 1994. Morgan Kaufmann.
- [Gordon *et al* 79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

- [Goren 86] Charles H. Goren. *Goren's New Bridge Complete*. Century Hutchinson Limited, 1986.
- [Grabiner 94] David Grabiner. Personal communication, September 1994.
- [Green 95] Stephen Green. Drawing inferences from bridge bidding. 4th Year Project Report, DAI, 1995.
- [Hsu *et al* 90] F. Hsu, T. Anantharaman, M. Campbell, and A. Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, October 1990.
- [Kambhampati & Srivastava 95] S. Kambhampati and B. Srivastava. Universal classical planner: An algorithm for unifying state-space and plan-space planning. Technical Report CSE TR 94-002, Arizona State University, 1995. To appear in Proc. 3rd European Planning Workshop (EWSP-95).
- [Kambhampati 92] S. Kambhampati. Multi-contributor causal structures for planning: A formalisation and evaluation. Technical Report CS TR-92-019, Arizona State University, 1992.
- [Kambhampati 94] Subbarao Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In *Proceedings of Principles of Knowledge Representation and Reasoning*, pages 329–340, May 1994.
- [Kambhampati 95] S. Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, 6(1), January 1995.
- [Kambhampati *et al* 95] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76:167–238, July 1995.
- [Khemani 94] Deepak Khemani. Planning with thematic actions. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 287–292, Menlo Park, CA, 1994. AAAI Press.
- [Kibler & Schwamb 92] D. Kibler and K. Schwamb. Complete contingency planners. Technical Report UCI TR 92-94, University of California, Irvine, 1992.
- [Knoblock 95] Craig Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Confer-*

- ence on Artificial Intelligence*, pages 1686–1693, Montreal, 1995. IJCAI.
- [Korf 89] Richard E. Korf. Generalized game trees. In *Proceedings of the 11th IJCAI*, volume 1, pages 328–333, Detroit, Michigan, USA, 1989.
- [Kushmerick *et al* 95] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.
- [Lee & Mahajan 88] K. Lee and S. Mahajan. A pattern classification approach to evaluation function learning. *AI Journal*, 36:1–25, 1988.
- [Lee 94] Ray Lee, editor. *Reviews from Canadian Master Point*. 1992–1994. Available by anonymous ftp as the file /pub/Bridge/FAQ.ReviewsFromCanadianMasterPoint from the site arp.anu.edu.au.
- [Levy 89] D.N.L. Levy. The million pound bridge program. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 95–103. Ellis Horwood, 1989.
- [Lindelöf 83] E.T. Lindelöf. *COBRA: The Computer-Designed Bidding System*. Victor Gollancz Ltd, London, 1983. A volume in the Master Bridge Series (series editor Kelsey, H.W.), ISBN 0-575-02987-0, available from IPBM, 455 Alfreton Road, Nottingham NG7 5LX, England, Phone: +44 602 422615.
- [Lopatin 92] A. Lopatin. Two combinatorial problems in programming bridge game. *Computer Olympiad*, 1992.
- [Love 59] C.E. Love. *Bridge Squeezes Complete*. Dover, 1959.
- [Lowe 91] Helen Lowe. Extending the proof plan methodology to computer configuration problems. *Artificial Intelligence Applications Journal*, 5(3), 1991. Also available from Edinburgh as DAI Research Paper 537.
- [Luce & Raiffa 57] R. Duncan Luce and Howard Raiffa. *Games and Decisions—Introduction and Critical Survey*. Wiley, New York, 1957.
- [Lustig 85] R. Lustig. Hibrid: A hierarchically designed bridge playing program. Unpublished M.Sc. thesis, Weizmann Inst. of Science, Rehovot, Israel, 1985.

- [MacLeod 91] J. MacLeod. Microbridge — a computer developed approach to bidding. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 81–87. Ellis Horwood, 1991.
- [Manley 94] B. Manley. Bridge software: Moving forwards or backwards? *The Bulletin, American Contract Bridge League*, October 1994.
- [McAllester & Rosenblitt 91] D. McAllester and D. Rosenblitt. Systematic non-linear planning. In *Proceedings of the 9th AAAI*, pages 634–639, Anaheim, CA, 1991. AAAI.
- [McDermott 91] D. McDermott. Regression planning. *International Journal of Intelligent Systems*, 6(4):357–416, 1991. Also available as Yale TR YALEU/CSD/RR 752.
- [Minton *et al* 91] S. Minton, J. Bresina, and M. Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of IJCAI-91*, 1991.
- [Napjus 69] C.N. Napjus. *Declarer: A Learning Bridge-Playing Program which Generalizes and Infers*. Unpublished PhD thesis, University of Washington, 1969.
- [Nash 51] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 2(54):286–295, 1951.
- [Nau *et al* 88] Dana Nau, Paul Purdom, and Chun-Hung Tzeng. An evaluation of two alternatives to minimax. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 505–509. Elsevier Science Publishers B.V., The Netherlands, 1988.
- [Nau *et al* 91] D.S. Nau, Q. Yang, and J. Hendler. Optimization of multiple-goal plans with limited interaction. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 160–165. Morgan Kauffman Publishers, Inc, November 1991.
- [Newell & Simon 76] A. Newell and H. Simon. Computer science as empirical enquiry: Symbols and search. In *Communications of the Association for Computing Machinery 19*. Association for Computing Machinery, Inc., 1976.
- [Nygate & Sterling 90] Y. Nygate and L. Sterling. Python: An expert squeezer. *The Journal of Logic Programming*, 8(1 and 2), January/March 1990.

- [Nygate & Sterling 93] Y. Nygate and L. Sterling. Aspen — designing complex knowledge based systems. In *10th Israeli Symposium on Artificial Intelligence*, pages 51–60, 1993.
- [Nygate 84] Y. Nygate. Python: A bridge expert on squeezes. Unpublished M.Sc. thesis, Weizmann Inst. of Science, Rehovot, Israel, 1984.
- [Parlett 91] David Parlett. *A History of Card Games*. Oxford University Press, 1991.
- [Pearl 81] Judea Pearl. Heuristic search theory: Survey of recent results. In *Proceedings of IJCAI-81*, volume 1, pages 554–562, Vancouver, Canada, 1981.
- [Pearl 83] Judea Pearl. On the nature of pathology in game searching. *Artificial Intelligence*, 20:427–453, 1983.
- [Pearl 84] Judea Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Publishing Company, 1984.
- [Pednault 88] E P D Pednault. Extending conventional planning techniques to handle actions with context-dependent effects. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 55–59, St Paul, MN, 1988. AAAI.
- [Pednault 91] E P D Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 240–245, Sydney, Australia, 1991. IJCAI.
- [Penberthy & Weld 92] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, 1992.
- [Peot & Smith 92] Mark A Peot and David E Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197, College Park, Maryland, 1992. Morgan Kaufmann.
- [Pryor & Collins 96] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, To appear, 1996.
- [Pryor 95] Louise Pryor. Decisions, decisions: Knowledge goals in planning. In J Hallam, editor, *Hybrid Problems, Hybrid Solutions (Proceedings of AISB-95)*,

- Frontiers in Artificial Intelligence and Applications, pages 181–192. IOS Press, Amsterdam, 1995.
- [Quinlan 79] J.R. Quinlan. A knowledge-based system for locating missing high cards in bridge. In *Proceedings of IJCAI-79*, pages 705–710, Tokyo, 1979.
- [Reese & Bird 83] Terence Reese and David Bird. *Bridge, the Modern Game*. Faber and Faber Ltd, 1983.
- [Reese & Dormer 91] Terence Reese and T. Dormer. *The Play of the Cards*. Robert Hale Limited, 1991.
- [Rosenbloom 82] P.S. Rosenbloom. A world-championship level othello program. *AI Journal*, 19:279–320, 1982.
- [Sacerdoti 74] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.
- [Sacerdoti 75] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of IJCAI-75*, pages 206–214, 1975.
- [Samuel 59] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–220, July 1959.
- [Schaeffer et al 92] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53:273–289, 1992.
- [Schaeffer et al 93a] J. Schaeffer, P. Lu, D. Szafron, and R. Lake. A re-examination of brute-force search. In *Games: Planning and Learning, Papers from the 1993 Fall Symposium*, pages 51–58, AAAI Press, 1993.
- [Schaeffer et al 93b] J. Schaeffer, N. Treloar, P. Lu, and R. Lake. Man versus machine for the world checkers championship. *AI Magazine*, 14(2):28–35, 1993.
- [Schoppers 87] M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. *Proceedings of IJCAI-87*, 2:1039–1046, 1987.
- [Shannon 50] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.
- [Sime 92] Iain Sime. Saitek Pro Bridge 500—a laptop bridge computer. *SBU News, The Official Magazine of the Scottish Bridge Union*, pages 5–6, November 1992. Issue Number 10.

- [Simon 83] H.A. Simon. Search and reasoning in problem solving. *Artificial Intelligence*, 21:7–29, 1983.
- [Smith & Nau 93] S.J.J. Smith and D.S. Nau. Strategic planning for imperfect information games. In *Games: Planning and Learning, Papers from the 1993 Fall Symposium*, pages 84–91, AAAI Press, 1993.
- [Smith & Nau 94] S.J.J. Smith and D.S. Nau. An analysis of forward pruning. In *Proceedings of AAAI-94*, pages 1386–1391, 1994.
- [Stanier 75] A. Stanier. Bribip: A bridge bidding program. In *Proceedings of the 4th IJCAI*, Tbilisi, USSR, 1975.
- [Stanier 76] A. Stanier. Planning to make tricks at bridge. In *Proceedings of the AISB Summer Conference*, pages 256–265, 1976. Also in D. Levy, editor, *Computer Games II*, Springer Verlag, New York, 1988.
- [Sussman 73] G. J. Sussman. A computational model of skill acquisition. Technical Report 297, MIT, August 1973. Also American Elsevier, New York, 1975.
- [Tate 75] A. Tate. Interacting goals and their use. *Proceedings of IJCAI-75*, pages 215–218, 1975.
- [Tate 77] A. Tate. Generating project networks. *Proceedings of IJCAI-77*, 1977.
- [Tesauro & Sejnowski 89] G. Tesauro and T.J. Sejnowski. A parallel network that learns to play backgammon. *AI Journal*, 39:357–390, 1989.
- [Tesauro 94] G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.
- [Throop 83] T. Throop. *Computer Bridge*. Hayden, 1983.
- [Uiterwijk *et al* 89] J.W.H.M. Uiterwijk, H.J. van den Herik, and L.V. Allis. A knowledge based approach to connect-four. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 113–133. Ellis Horwood, 1989.
- [van Harmelen 89] F. van Harmelen. The CLAM proof planner, user manual and programmer manual: version 1.4. Technical Paper TP-4, DAI, 1989.

- [Vasconcelos 95] W. W. Vasconcelos. *Extracting, Organising, Designing and Reusing Prolog Programming Techniques*. Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, August 1995.
- [von Neumann & Morgenstern 44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, 1944.
- [Warren 76] D.H.D. Warren. Warplan: A system for generating plans. Technical Report Memo 76, Computational Logic Dept., School of Artificial Intelligence, University of Edinburgh, 1976.
- [Wasserman 70] A. Wasserman. Realisation of a skillful bridge bidding program. In *Proceedings of the Fall Joint Computer Conference — AFIPS*, Houston, Texas, 1970.
- [Wheen 89] R. Wheen. Solving double dummy bridge problems by exhaustive search. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence — The First Computer Olympiad*, pages 89–94. Ellis Horwood, 1989.
- [Wilkins 84] D.E. Wilkins. Domain-independent planning: Representations and plan generation. *Artificial Intelligence*, 22:269–301, April 1984.
- [Wilkins 88] D.E. Wilkins. *Practical Planning*. Morgan Kaufmann Publishers, Inc, 1988.
- [Wilkins 89] D.E. Wilkins. Can AI planners solve practical problems? Technical Report Technical Note 468, SRI International, 1989.

Index

Symbols

- 8 Puzzle 368
- 8 Queens Problem 40

A

- action-ordering representation 34
- actions 33, 367
- agents 367
- anteriority 112
- anytime algorithm 249
- ASPEN 20
- auction *see* bidding
- auxiliary constraints ... *see* constraints
- average propagation algorithm 140

B

- Backgammon
 - BKG9.8 29
 - TD-GAMMON 29
- behavioural strategies... *see* strategies
- best defence model 116–118, 271, 283, 367
- best-case profiles *see* profiles
- bidding
 - computer
 - COBRA 16–18
 - rule-based 12–14
 - with look-ahead 15–16
 - making inferences from .. 14–15, 27
 - the auction 7
 - the contract 7
- BILL *see* Othello
- binding constraints *see* constraints
- Bridge King 19
- BURIDAN 38

C

- C-conjunctions 167–168
 - collecting 174
 - probabilities from 174–181
 - redundancy in 168–170

- text explanations from 244, 310–318

- card combinations .. 25, 65, 74–82, 368
- cashing 74–75, 191, 368
- CASSANDRA 38
- C-BURIDAN 38
- checkers
 - CHINOOK 29–31, 65, 290
 - Samuel's program 28
- chess
 - DEEP THOUGHT 30
- CLAM 86, 89, 103
- clichés 190, 231, 283
- CNLP 38
- COBRA *see* bidding, computer
- communication 191, 249
- Computer Games Olympiad ... 19, 243
- conflict resolution 52
- Connect-Four 28
- constraints
 - auxiliary 44–46
 - binding 42–44
 - interval preservation 44
 - monotonicity of 46
 - ordering 42–44
 - point truth 45
- context-dependency ... 37, 87, 196, 368
- contingencies 38, 189
- contingency planning 38, 189
- contract *see* bidding
- contributor protection 50
- control 81
- cracking a game 21, 29, 368
- critical cards 3, 85–86, 368

D

- declarer 7
- DEEP THOUGHT *see* chess
- defenders 7
- direct signalling 114

discards 84, 91, 226, 237, 308, 368
 discovery plays 136
 domain rules 87, 96, 239, 285
 dominance 251
 double-dummy Bridge 26, 66–68,
 126–128
 doublethink 152
 doubletons 314
 ducking 80–81, 200, 368
 dummy 7
 duplicate Bridge 253, 368
 dynamic programming 97

E

strategy fusion 128–137, 283
 entries 193
 equilibrium points 110
 exclusion sets 152
 exhaustive minimaxing *see* mini-
 maxing
 exhaustive strategy minimisation . 119–
 120, 283
 extended operator representation . 190,
 368
 extensive form of a game 107–109

F

false-carding 271
 filter conditions 48
 finessing 75–80, 369
 examples 23, 88, 134, 142, 194,
 264–266
 following suit 7, 237

G

generate-and-test 39
 global profiles *see* profiles
 Go 31
 goals
 intermediate 48
 maintenance 48
 of attainment 48
 GPS 36
 graphical interface 92–94
 ground linearisations 44
 safe 47
 ground operator sequences... 33, 43–44
 guards 235

H

HACKER 36
 history lists 97–103
 HTN planners 38

I

IAGO *see* Othello
 incidence calculus 176
 information sets 108, 369
 inter-suit tactics 233–245, 288, 369
 interleaving 186, 283
 as refinement search 205
 interleaving constraint 218
 intermediate goals *see* goals
 intermediate techniques 190
 INTERPLAN 37
 interpreter algorithm 161–164, 369
 interval preservation constraints *see*
 constraints
 interval protection 50
 inverted signalling 114
 IPC *see* constraints, interval
 preservation
 IPEM 39
 irredundancy 251

L

leads 191
 legal play sequences 53–69, 369
 lines of play 3, 19, 164, 187, 369
 losers 257

M

maintenance goals *see* goals
 majorant games 111
 mathematical programming 117
 MEA *see* means-ends analysis
 means-ends analysis 36
 menaces
 single 235
 two-card 235
 methods 86–89, 369
 million pound bet 7, 126
 minimaxing
 exhaustive 127
 naïve 150
 payoff reduction 158
 repeated 127
 Shannon's introduction of 28

the minimax theorem 110–112
 with incomplete information ... 22,
 121–122
 minorant games 111
 mixed strategies *see* strategies
 monotonicity *see* constraints
 MTC *see* truth criterion

N

naïve minimaxing *see* minimaxing
 NOAH 37
 non-cooperative games 110
 non-linear planners 34
 non-locality ... **137–143**, 151, 283, 289,
 370
 heuristics to control 181–183
 NONLIN 37, 51
 normal form of a game 109–110

O

oddthink 153–155
 operators 33, 370
 O-PLAN 38
 ordering constraints *see* constraints
 Othello
 BILL 29
 IAGO 29
 OYSTER 354

P

partial plans 35, **42**
 partially ordered plans 34
 payoff profiles *see* profiles
 payoff reduction 155–161
 payoff reduction minimaxing *see*
 minimaxing
 payoffs
 8-tuples 201, 258
 definition 107, 370
 payoff function 107
 vectors 146
 PEDESTAL 38
 perfect information 113
 plan objects 247
 plan-space planning 35
 play of a game 107
 PLINTH 38
 point truth constraints . *see* constraints
 Poker 113

pre-ordering refinements 52
 preference 226
 preliminary 112
 probability profiles *see* profiles
 product propagation algorithm 140
 profiles
 best-case 253–259
 global 247
 payoff 162
 probability 163
 resource 196–203
 shift operation on 250
 proof-planning **71–74**, 284, 370
 motivation for using 2
 protection strategies 50, 228
 PROTEUS 316
 PTC *see* constraints, point truth
 pure strategies *see* strategies
 PYTHON *see* squeezing

Q

QA *see* Question Answering
 qualitative representation 146
 Qubic 28
 Question Answering 37

R

reactive planning 39
 refinement search **39–42**, 203, 283
 interleaving as 205
 repeated minimaxing .. *see* minimaxing
 resource profiles *see* profiles
 resources **190**
 rubber Bridge 253, 370
 ruffing
 definition 7, 370
 examples 24

S

safe ground linearisations .. *see* ground
 linearisations
 SAGE 39
 search space size
 of Bridge 53–69
 of other games 64
 secondary preconditions 38, 49
 SENSP 38
 sequence tactic 81–82, 192, 371
 shape of a hand 55

shape-list algorithm 255
 shape-lists 179–181
 signalling 84, **114**
 single menace *see* menaces
 single-suit planning 25, 89–92
 singletons 314
 SIPE 37, 87
 Situated Control Rules 39
 SNLP 38, 51
 solution candidates 39
 solution constructor function 40, 48
 in Bridge 220–231, 237–242
 solving a game **21**, 29, 371
 squeezing
 examples 23, 234–236
 incorporating in interleaver 236
 PYTHON 26, 242
 state-space
 planning 35
 representation 34
 strategies 109, 371
 behavioural 367
 mixed 112, 290, 370
 pure 112, 290, 370
 STRIPS 36, 87
 subsumption **173**, 202, 241
 Sussman anomaly 36, 189

T

tactics 371
 in FINESSE 74–83
 in other Bridge systems 20–21
 in theorem proving 72
 tenaces 77
 TIGNUM 21, 24
 top tricks 22, 247
 total ordering 38, 52
 tractability refinements **49**, 50–52,
 206, 283
 tricks 7
 triplethink 153
 trump suit 7, 371
 truth criterion **37**, 190, 371
 TWEAK 37, 51
 two-card menace *see* menaces

U

UCPOP 38, 51
 unambiguous ordering 52

universal plans 39

V

voids **7**, 56, 83, 224, 314, 371
 vulnerability 253, 371

W

WARPLAN 37
 weak dominance 251
 worlds 105, **122**, 372

Z

zero-sum games 110