

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Guid

**Znanje in preiskovanje
pri človeškem in računalniškem
reševanju problemov**

DOKTORSKA DISERTACIJA

Ljubljana, 2010

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Guid

**Znanje in preiskovanje
pri človeškem in računalniškem
reševanju problemov**

DOKTORSKA DISERTACIJA

Mentor: Akad. Prof. Dr. Ivan Bratko

Ljubljana, 2010

University of Ljubljana
Faculty of Computer and Information Science

Matej Guid

**Search and Knowledge for Human
and Machine Problem Solving**

DOCTORAL DISSERTATION

Supervisor: Acad. Prof. Dr. Ivan Bratko

Ljubljana, 2010

Abstract

In Artificial Intelligence (AI), there exist formalised approaches and algorithms for general problem solving. These approaches address problems that require combinatorial search among alternatives, such as planning, scheduling, or playing of games like chess. In these approaches, problems are typically represented by various kinds of graphs, and problem solving corresponds to searching such graphs. Due to their combinatorial complexity, these problems are solved by heuristic search methods where problem-specific heuristics represent the solver's knowledge about a concrete problem domain. Thus such computer-based approaches to problem solving roughly consist of two components: *search* among alternatives, and problem-specific *knowledge*. Computer methods of heuristic search are also a good model of human problem solving. In human problem solving, however, these two components take very different dimensions compared with machine problem solving. A human expert typically uses much richer domain-specific knowledge whereas the computer has the advantage of incomparably faster search compared to the human.

The thesis presents some novel aspects on the comparison and combination of search and knowledge in human and machine problem solving, in particular with respect to possibilities of developing heuristic-search methods for evaluating and improving problem-solving performance. Among others, the following scientific questions are addressed. How can a computer be used to assess a human's problem-solving performance? How can a machine problem solving model be used to assess the difficulty of a given set of problems for a human? How can machine problem solving be used in tutoring, for teaching a human to solve problems in a given problem domain? How can knowledge represented in a form suitable for the computer, be transformed into a form that can be understood and used by a human? In this thesis we explore these questions in the framework of human and computer game playing, and use the game of chess as the experimental domain.

In Part I of the thesis, "Search and Knowledge for Estimating Human Problem Solving Performance," we demonstrate that heuristic-search based programs can be useful estimators of human problem-solving performance. We introduced a novel method, based on computer heuristic search, for evaluating problem-solving performance in chess (with possible extensions to other games), and provided an analysis of appropriateness of this method. Experimental results and theoretical explanations were provided to show that, in order to obtain a sensible ranking of the chess players

using our method, it is not necessary to use a computer that is stronger than the chess-players themselves. We also designed a heuristic-search based method for assessing the average difficulty of a given set of chess positions (problem situations).

In Part II, “Search and Knowledge for Improving Human Problem Solving Performance,” we presented a novel, heuristic-search based approach to automated generation of human understandable commenting of decisions in chess. We also demonstrated a novel approach to the formalization of complex patterns for the purpose of annotating chess games using computers. Finally, we introduced a procedure for semi-automatic synthesis of knowledge suitable for teaching how to solve problems in a given domain. We verified appropriateness of this procedure in a case study where we applied it to obtain human-understandable textbook instructions for teaching a difficult chess endgame.

Part III, “On The Nature of Heuristic Search in Computer Game Playing,” aims at improving the understanding of properties of heuristic search and consequences of the interaction between search and knowledge that typically occurs in both human and machine problem solving. *Monotonicity property* of heuristic evaluation functions for games was revisited. Namely, that backed-up values of the nodes in the search space have to tend to approach monotonically to the terminal values of the problem state space with the depth of search. We pointed out that backed-up heuristic values therefore do not approximate some unknown “true” or “ideal” heuristic values with increasing depth of search, in contrast to what is generally assumed in the literature. We also discussed some of possible impacts of this property on the theory of game playing, and pointed out that heuristic evaluations obtained by searching to different search depths are not directly comparable, in contrast to what is generally assumed both in literature and in practical applications. Finally, we studied experimentally factors which influence the behavior of diminishing returns with increased search. Empirical proof was provided that the rate of changed decisions that arise from search to different depths depends on (1) the quality of knowledge in evaluation functions, and (2) the value of a node in the search space.

Keywords

artificial intelligence, heuristic search, problem solving; heuristic evaluation functions, estimating problem-solving performance, intelligent tutoring, intelligent annotating, expert systems, knowledge elicitation; game playing, chess, computer chess

Povzetek

V umetni inteligenci obstajajo formalizirani in algoritmizirani pristopi k reševanju problemov, ki so po svoji naravi kombinatorični, kot je npr. načrtovanje operacij ali igranje miselnih iger. V teh pristopih so problemi predstavljeni s problemskim prostorom, ki je značilno neke vrste graf, reševanju problema pa ustreza preiskovanje ustreznega grafa. Zaradi kombinatorične zahtevnosti se ti problemi rešujejo s hevrističnimi metodami preiskovanja, pri katerih problemu specifične hevristike predstavljajo znanje o konkretni problemski domeni. Tako ti računalniški pristopi v umetni inteligenci v grobem vsebujejo dve komponenti: specifično *znanje* o problemu ter *preiskovanje* med alternativami. Metode hevrističnega reševanja problemov so tudi dober model človeškega reševanja problemov, ki prav tako odraža ti dve komponenti. Seveda pa sta pri računalnikih in ljudeh ti dve komponenti zelo različno zastopani. Človek – ekspert tipično uporablja veliko bogatejše znanje o samem problemu, medtem ko je prednost računalnika v neprimerno hitrejšem preiskovanju.

Doktorske disertacija obravnava možnosti razvoja metod, temelječih na hevrističnem preiskovanju, za ocenjevanje in izboljševanje uspešnosti pri človeškem in računalniškem reševanju problemov. Vsebuje tudi prispevke k splošnemu razumevanju lastnosti hevrističnega preiskovanja ter posledic interakcije med znanjem in preiskovanjem, tipično prisotne pri reševanju problemov, tako pri ljudeh kot pri računalnikih. Med drugim smo si zastavili naslednja vprašanja. Kako uporabiti računalnik pri ocenjevanju uspešnosti človekovega reševanja problemov? Kako z modelom računalniškega reševanja oceniti, kako težavni so konkretni dani problemi za človeka? Kako bi lahko računalniško reševanje problema uporabili za poučevanje človeka o reševanju problemov v dani problemski domeni? Kako transformirati znanje, izraženo v obliki, primerni za računalnik, v obliko, ki jo razume in lahko uporabi človek? V doktorski disertaciji so ta vprašanja naslovljena v ogrodju človeškega in računalniškega igranja iger, šah pa je uporabljen kot raziskovalna domena.

V prvem delu disertacije, “Ocenjevanje uspešnosti ljudi pri reševanju problemov”, smo pokazali, da so programi, ki temeljijo na hevrističnem preiskovanju, lahko uspešni ocenjevalci uspešnosti ljudi pri reševanju problemov. Predstavili smo novo metodo, temelječo na hevrističnem preiskovanju, za ocenjevanje uspešnosti reševanja problemov v šahu (z možnostjo razširitve na ostale igre), in pokazali verodostojnost te metode. Eksperimentalni rezultati in teoretične razlage so podprle našo tezo, da lahko računalniški program z uporabo naše metode ustrezno razvrsti šahiste glede

na njihovo uspešnost pri reševanju problemov, četudi je po šahovski moči slabši od njih. Prav tako smo razvili novo metodo, ki temelji na računalniškem hevrističnem preiskovanju, za ocenjevanje povprečne težavnosti množice šahovskih pozicij (problemskih situacij) za človeka.

V drugem delu disertacije, "Izboljševanje uspešnosti ljudi pri reševanju problemov", smo najprej predstavili nov pristop, temelječ na računalniškem hevrističnem preiskovanju, k avtomatskemu in hkrati človeku razumljivemu komentiranju odločitev v šahu. Razvili smo nov pristop k formalizaciji kompleksnih vzorcev za namen računalniškega komentiranja šahovskih partij. Predstavili smo tudi nov pristop k polavtomatskemu pridobivanju človeku razumljivega znanja, primerne za poučevanje reševanja problemov v dani problemski domeni. Ustreznost tega pristopa smo preverili s študijo, kjer smo z njegovo uporabo pridobili človeku razumljiva navodila za poučevanje težavne šahovske končnice.

Tretji del disertacije, "O naravi hevrističnega preiskovanja pri računalniškem igranju iger", stremi k izboljšanju razumevanja lastnosti hevrističnega preiskovanja in posledic interakcije med znanjem in preiskovanjem, tipično prisotne pri reševanju problemov, tako pri ljudeh kot pri računalnikih. Raziskali smo lastnost *monotonosti* hevrističnih ocenjevalnih funkcij pri igranju iger: z naraščajočo globino preiskovanja morajo vzvratne ocene vozlišč težiti k monotonemu približevanju končnim vrednostim v prostoru preiskovanja. Vzvratne ocene torej ne aproksimirajo nekih "resničnih" vrednosti ali "idealnih" hevrističnih vrednosti, kar se sicer v literaturi na splošno predpostavlja. Razložili smo nekaj možnih vplivov lastnosti monotonosti na teorijo igranja iger. Pokazali smo, da hevristične ocene, pridobljene pri različnih globinah iskanja, niso primerljive med seboj, kot je sicer splošno predpostavljeno tako v literaturi kot v praktičnih aplikacijah. V nadaljevanju smo izvedli eksperimentalno študijo v zvezi z dejavniki, ki vplivajo na spreminjanje odločitev z globino preiskovanja. Empirično smo dokazali novi ugotovitvi, da je pogostost razlik v odločitvah, ki temeljijo na različnih globinah preiskovanja, odvisna od (1) kvalitete hevrističnega znanja v ocenjevalni funkciji in (2) vrednosti vozlišča v preiskovalnem prostoru.

Ključne besede

umetna inteligenca, hevristično preiskovanje, reševanje problemov; hevristične ocenjevalne funkcije, ocenjevanje uspešnosti pri reševanju problemov, inteligentno poučevanje, inteligentno komentiranje, ekspertni sistemi, zajemanje znanja; igre, šah

Acknowledgements

First and foremost, I wish to express my gratitude to professor Ivan Bratko, my supervisor, for his best efforts to teach me scientific thinking and writing, for showing me how to carry out my research at high standards, for sharing his experience and offering great pieces of advice, and for the privilege of working in a relaxed and stimulating environment. I am especially grateful for the opportunity to use chess, my favorite game, as the research domain – in this way it was a true joy to carry out my research and although I did my best to carry it out with great responsibility, I almost considered my work as a hobby, thus my interest for Artificial Intelligence grew daily. Thanks to Ivan’s great interest in chess and his understanding of the many intricacies of the game, I was truly lucky to have a supervisor that few places on Earth provide!

Moreover, I would like to thank to all the members of Artificial Intelligence Laboratory for providing an excellent working environment. My special thanks goes to my friends Saša Sadikov, Martin Možina, Jana Krivec, and Vida Groznik, the co-authors of several papers we worked on together, for the great atmosphere that we created, keeping our joint work both productive and fun. Starting from this point, I should not forget to thank the staff of Korner Pub, which offered us a creative environment where many new ideas were born. Of course, ideas alone were not enough and quite often working until early morning hours (or sometimes even later) was on the agenda – at this point I would like to thank the Faculty of Computer and Information Science for giving me at disposal an enormous computing power in the form of about a hundred computers. My special thanks also goes to Aritz Pérez Martínez, the co-author of my “hat-trick” ICGA Journal paper (*i.e.*, the third paper in my favorite scientific journal during three consecutive years – as at the time of this writing the 2010 FIFA World Cup is taking place, please allow me to use the above expression), for his collaboration during his three-month visit in our lab.

I would also like to express my sincere gratitude to professor Jaap van den Herik, the editor of International Computer Games Association (ICGA) Journal and also a member of my thesis evaluation committee, for many instructive points behind his truly extensive comments on an earlier version of this dissertation.

Other people whom I would like to thank for their comments and feedback include Guy Haworth, Renze Steenhuisen, Blaž Zupan, Janez Demšar, Lan Umek, Gorazd Lampič, Goran Bobojevič and Matej Marinč. I am also very grateful to many

other people – most of whom I never met – for the large number of interesting points among their ample feedback and for their interest in my research.

My deepest gratitude also to the others who have been dear friends during my Ph.D. studies. Above all, I thank my family for the patience and support during these five years. This dissertation is dedicated to my parents Boris and Nuška, my brother Rok, my sister Anja, my lovely wife Sabina, and my two beautiful daughters Taja and Ana.

Contents

1	Introduction	7
1.1	Problem Solving and Heuristic Search	8
1.2	Search and Knowledge	8
1.3	Game Playing as a Platform for AI Research	9
1.4	Chess as a Research Domain	10
1.5	Related Work	11
1.5.1	Human Problem Solving and Integrated Cognitive Architec- tures	12
1.5.2	Artificial Intelligence in Education	14
1.6	Research Summary	17
1.7	Thesis Overview	18
1.8	Contributions to Science	20
I	Search and Knowledge for Estimating Human Problem Solving Performance	23
2	Computer Analysis of World Chess Champions	25
2.1	Can Heuristic Search be Useful for Estimating Problem Solving Per- formance?	26
2.2	Determining Best Chess Player in History	27
2.3	Obtaining Data for the Analysis	29
2.4	Three Criteria for Estimating Performance	30
2.4.1	Basic Criterion	30
2.4.2	Blunder Criterion	31
2.4.3	Best Moves Criterion	32
2.5	Player’s Performance w.r.t. Difficulty of Positions	34

2.6	Results of a Computer Analysis	35
2.6.1	Results According to the Basic Criterion	35
2.6.2	Blunder-Rate Measurements	36
2.6.3	Bringing the Champions to a “Common Denominator”	36
3	Credibility of a Heuristic-Search Based Estimator	41
3.1	Trustworthiness of CRAFTY’s Analysis of World Chess Champions	42
3.2	Variation of Rankings with Search Depth	43
3.3	Robustness of Rankings w.r.t. Sample Size	44
3.3.1	Number of Positions for Analysis	45
3.3.2	Stability of the Rankings with Search Depth	50
3.4	A Simple Probabilistic Model of Ranking by an Imperfect Referee	52
3.5	A Model of Estimators of Different Strengths	53
3.5.1	Variance of Players’ Scores and Rankings with Search Depth	55
3.6	Using Other Programs as Estimators	56
4	Assessing Difficulty of Problem Solving Tasks	61
4.1	Difficulty Measurements in the WCC matches	64
II	Search and Knowledge for Improving Human Problem Solving Performance	67
5	A Heuristic-Search Based Annotator	69
5.1	Automatic Annotation of Chess Games	70
5.1.1	Related Work	70
5.2	The Annotating System Design	72
5.3	Our Approach to Automatic Annotation	73
5.3.1	Commenting on Good Characteristics	74
5.3.2	Commenting on Bad Characteristics	76
5.4	The Rule-Based Expert System	78
5.4.1	Illustrative Example	78
5.4.2	Possibilities for Manually Changing the Obtained Rules	82
5.5	Final Remarks	83
6	Obtaining Knowledge for a Heuristic-Search Based Program	85
6.1	Knowledge Acquisition for Chess Annotation	86

6.2	Positional Features for Annotating Chess Games	87
6.2.1	The Static Nature of Positional Features	89
6.3	Application of Machine Learning Techniques	89
6.3.1	The Learning Data Set	90
6.3.2	Using Ordinary Machine Learning Methods	91
6.3.3	Argument Based Machine Learning	92
6.4	Knowledge Elicitation Process	94
6.5	Assessment and Discussion	97
7	Deriving Concepts and Strategies from Chess Tablebases	101
7.1	Learning from Perfect Information	102
7.2	Semi-Automatically Derived Instructions for KBNK endgame . . .	103
7.2.1	The Hierarchical Model of Ordered Set of Rules	108
7.2.2	Generating Example Games	110
7.3	The Process of Synthesizing Instructions	110
7.3.1	Basic Description of Our Approach	110
7.3.2	Obtaining Knowledge from Domain Expert	111
7.3.3	Strategic Goal-Based Rules	111
7.3.4	Allowing Non-Optimal Play	112
7.3.5	Hierarchy of Goals	113
7.3.6	Constructing Human-Friendly Instructions from Semi-Automatically Generated Rules	113
7.3.7	Demonstration of Interaction between Computer and Domain Expert in KBNK	114
7.4	Discussion and Evaluation	117
7.5	Final Remarks	120
	III On The Nature of Heuristic Search in Computer Game Playing	121
8	Monotonicity Property of Heuristic Evaluation Functions	123
8.1	Heuristic Evaluation in Game Playing	124
8.1.1	What are “Ideal” Heuristic Values?	125
8.1.2	Direction Oriented Play	127
8.1.3	Our Theoretical Model	129

8.2	Experimental Design	132
8.3	Experimental Results	133
8.4	Heuristic Evaluation Functions and Probability of Winning	141
8.5	Possible Impacts on Theory and Practice	144
8.5.1	Searching to Variable Depths Revisited	144
8.5.2	Expectations of Decision Changes with Deeper Search	148
8.5.3	Detecting Fortresses in Chess	148
9	Factors Affecting Diminishing Returns for Searching Deeper	151
9.1	Go-Deep Experiments and Diminishing Returns	151
9.2	Experimental Design	154
9.3	Diminishing Returns and Values of Positions	155
9.3.1	CRAFTY Goes Deep	155
9.3.2	RYBKA Goes Deep	157
9.4	Diminishing Returns and Quality of Evaluation Function	160
9.5	Diminishing Returns and Phase of the Game	162
10	Conclusions	165
10.1	Critical Analysis and Open Questions	165
10.2	Search and Knowledge for Estimating Human Problem Solving Performance	167
10.3	Search and Knowledge for Improving Human Problem Solving Performance	170
10.4	On the Nature of Heuristic Search in Computer Game Playing	173
A	Razširjeni povzetek v slovenskem jeziku (Extended Abstract in Slovene Language)	177
A.1	Uvod	179
A.1.1	Reševanje problemov in hevristično preiskovanje	180
A.1.2	Preiskovanje in znanje	180
A.1.3	Igranje iger kot platforma za raziskave v umetni inteligenci	181
A.1.4	Šah kot raziskovalna domena	182
A.1.5	Ocenjevanje in izboljševanje uspešnosti pri reševanju problemov	184

A.2	Ocenjevanje uspešnosti ljudi pri reševanju problemov	186
A.2.1	Računalniška primerjava svetovnih šahovskih prvakov	186
A.2.2	Verodostojnost ocenjevalca uspešnosti, temelječega na hevrističnem preiskovanju	192
A.2.3	Ocenjevanje težavnosti šahovskih pozicij	194
A.3	Izboljševanje uspešnosti ljudi pri reševanju problemov	197
A.3.1	Avtomatsko komentiranje šahovskih partij	198
A.3.2	Formalizacija kompleksnih vzorcev za namen komentiranja odločitev pri reševanju problemov	199
A.3.3	Polavtomatsko sintetiziranje človeku razumljivega znanja iz tabeliranih šahovskih baz	199
A.4	O naravi hevrističnega preiskovanja pri računalniškem igranju iger	200
A.4.1	Monotonost kot lastnost hevrističnih ocenjevalnih funkcij	201
A.4.2	Dejavniki, ki vplivajo na spreminjanje odločitev z globino preiskovanja	202
A.5	Prispevki k znanosti	204

Bibliography**209**

Chapter 1

Introduction

In Artificial Intelligence (AI), there exist formalized approaches and algorithms for general problem solving. These approaches address problems that require combinatorial search among alternatives, such as planning, scheduling, or playing of games like chess. In these approaches, problems are typically represented by various kinds of graphs, and problem solving corresponds to searching such graphs. Due to their combinatorial complexity, these problems are solved by heuristic search methods where problem-specific heuristics represent the solver's knowledge about a concrete problem domain. Thus such computer-based approaches in AI roughly consist of two components: *search* among alternatives, and problem-specific *knowledge*.

Computer methods of heuristic search are also a good model of human problem solving which also exhibits these two components: search and domain-specific knowledge. These two components, however, take very different dimensions in human problem solving compared with machine problem solving. A human expert typically uses much richer domain-specific knowledge whereas the computer has the advantage of incomparably faster search compared to the human. The well-known pioneering work on the modeling of computer problem solving with computer problem solving was carried out by A. Newell and H. Simon [NS72].

This thesis presents some novel aspects on the comparison and combination of search and knowledge in human and machine problem solving, in particular with respect to possibilities of developing heuristic-search methods for evaluating and improving problem-solving performance. It also aims at improving the understanding of properties of heuristic search and consequences of the interaction between search and knowledge that typically occurs in both human and machine problem solving.

In the introduction, we briefly introduce problem solving, heuristic search, and interaction between search and knowledge. Then suitability of the game-playing framework as a platform for research in AI, and appropriateness of the game of chess as a research domain is discussed. Finally, the major questions behind our research and contributions to science are stated.

1.1 Problem Solving and Heuristic Search

A person is confronted with a problem when he* wants something and does not know immediately what series of actions he can perform to get it [NS72]. In AI, a typical general scheme for representing problems is called state space. A state space is a graph of which the nodes correspond to problem situations, and a given problem is reduced to finding a path in this graph.

Graph searching in problem solving typically leads to the problem of combinatorial complexity due to the rapidly growing number of alternatives. To overcome this problem, heuristic search is widely used. For the nodes in the state space heuristic estimates are determined, indicating how promising nodes are with respect to reaching a goal node. The underlying idea is to perform search always from the most promising node among the candidate nodes [Bra00]. In general, heuristics stand for strategies that use information to control problem solving in human beings and machines [Pea84].

1.2 Search and Knowledge

Heuristic search implies usage of both search and knowledge. Knowledge can be used to guide the search and search may help to confirm the knowledge. A faster program can examine more nodes and therefore search deeper. This suggests a possible solution to solving various types of problems: obtain faster hardware. A different approach to the problem could draw on extensive knowledge to evaluate problem states accurately using less searching. Taken to the extreme, a program (or human) with deep enough understanding of the problem domain might not require any search at all [Sch86]. In chess, for example, human chess-players usually rely on their knowledge and experience with the game to perform less searching with position evaluations as

*For brevity we will use 'he' ('his') when 'he or she' ('his or her') is meant.

accurate as possible. Perfect knowledge about a domain renders search unnecessary and, likewise, exhaustive search obviates heuristic knowledge. In practice, a trade-off between search and knowledge is found somewhere in the middle, since neither extreme is feasible for interesting domains [JS99].

Search and knowledge are also fundamental components of expert systems. Experts systems, in general, contain problem-solving functions capable of using domain-specific knowledge, and this usually involves searching [Sch86; Bra00]. As knowledge is a key component of every intelligent computer system, obtaining knowledge is one of the perennial tasks of artificial intelligence. This process, called knowledge elicitation, is known to be a difficult task and thus a major bottleneck in building a knowledge base in expert systems [Fei03].

1.3 Game Playing as a Platform for AI Research

Most games of any interest cannot be played at an acceptable level without using domain knowledge, because the corresponding state space is too large to be searched completely in a reasonable amount of time. Consequently, they can neither be played by using knowledge nor search only. Moreover, ever since the beginning of artificial intelligence, game-playing provided a great platform for improving AI algorithms and methods. In games, the players (humans or computers) continuously deal with problems they have to solve. Therefore, game-playing was chosen as a suitable platform for the topics of this thesis.

In the usual game-theoretic framework, the state space is represented by a game tree. In the practice of computer game playing, only a part of complete game tree is generated, called a search tree, and a heuristic evaluation function is applied to terminal positions of the search tree. The heuristic evaluations of non-terminal positions are obtained by applying the minimax principle. That is, the estimates propagate up the search tree, determining the position values in the search tree. The so-called minimax search remains an essential component for programs that also incorporate a large amount of knowledge derived from a humanlike approach to understanding game states and move choices [Bea99].

Many experiments have been performed in game-playing programs that measure the benefits of improved knowledge and/or deeper search. In particular, chess has been a popular domain for these experiments. The explicit or implicit message of these works is that the results for chess are generalizable to other games [JS99].

1.4 Chess as a Research Domain

Newell and Simon [NS72] denote chess as a particularly attractive research domain for human problem solving for several reasons.

- Selecting a move in chess is generally acknowledged to be a difficult problem solving task.
- The vast amount of recorded experience makes it easy to evaluate the quality of a chess-playing program and compare it in detail with human players of different strength, different styles, and even different periods in the history of the game. Moreover, the protocols produced by a chess program can be compared with human protocols in the same game positions.
- The task has already been used in previous researches, particularly in the work by A. de Groot with human chess players [dG78].[†]
- The irregularity of the structure of chess gives the task some of the flavor of everyday, garden-variety problem solving that is absent from tasks like proving theorems or solving puzzles.

Schaeffer [Sch86] advocates that chess has many advantages as a domain for exploring some of the problems in artificial intelligence. A chess-program's performance is strongly tied to both search algorithms and its domain knowledge. Due to complexity of the game (around 10^{46} different positions are possible [Chi96]) perfect play is not feasible, so chess programs must have general knowledge that attempts to describe as many positions as possible. The result is that the knowledge is inexact (heuristic), and the program must make important quality-of-response versus effort-expended decisions. He also states the following advantages.

- The game is intellectually challenging; 200 years of intensive analysis has failed to make it any less interesting and did not exhaust its possibilities.
- The rules of chess are well-defined.
- Chess can be partitioned into manageable subsets; for example restricting the problem domain to endgame.

[†]Note that also after the year 1978, several other researchers from the field of psychology used chess as a research domain.

- Chess ratings are an accepted method for measuring performance. This is important in that improvements in the play of a chess program will be reflected in its ratings.
- There is a large body of chess knowledge to draw on.
- A great deal is known about humans and computers play chess.
- Many AI researchers know how to play chess and are capable of evaluating results from a chess program. For many other applications, the average researcher is unfamiliar with the domain and must rely on the opinions of others.

In the year 2010, we can add the following statements about the appropriateness of chess as our research domain.

- The strong chess programs are though opponents even to the strongest human grandmasters - already surpassing them in many aspects [Kas06].
- Complete tablebases [Tho86], indicating best moves for every position, exist for chess endgames (up to 6-pieces, including the kings).
- A user-friendly software to work with existing large databases containing millions of chess games is available and widely used.
- Chess-programs giving high quality decisions about best possible moves from a given position in a few minutes or sometimes even just in a few seconds are available and widely used.
- Chess is still a very popular game (or even more popular than in the past).

1.5 Related Work

The aim of this section is *not* to undertake a full-scale history of a related work in such a large scientific area such as Human and Machine Problem Solving, but merely to express a somewhat personal view, in particular with respect to the scope of this thesis, on the developments in the two highly related scientific fields where an interaction between human cognition and artificial intelligence also plays an important role.

1.5.1 Human Problem Solving and Integrated Cognitive Architectures

Newell and Simon [NS72] presented human cognition as an “Information Processing System” and defined its several different components, including human information processing capabilities such as short-term memory, long-term memory, external memory, perception, etc. They proposed an architecture for human cognition, suggesting that humans are essentially symbol manipulators that perform operations serially and represent knowledge as production rules, while solving problems by searching through a problem space with explicit representation of goals. Their proposed architecture is still perceived as a reasonable approximation of human cognition for the purpose of studying problem solving.

The work by Newell and Simon stimulated several researchers in psychology to aim towards various theories about human cognition. Newell [New90] introduced his view on a unified theory of cognition, that is set of mechanisms that account for all of human cognition, such as human memory, problem solving and planning, recognition and categorization, skill acquisition, and behavior in general. Such theory must explain, among other things, how intelligent organisms represent their knowledge, how they acquire knowledge, and how they operate flexibly according to their environment. Various theories for simulating and understanding human cognition were introduced by different authors [And93b; KWM97; LC06].

Based on theories of human cognition, several integrated cognitive architectures emerged, such as SOAR [LNR87], ACT-R [ABB⁺04], and ICARUS [LC06]. All three share many features of artificial intelligence, including means-end analysis for problem solving, symbolic representation of knowledge, inference based on production rules etc. While so far no cognitive architecture provided a full level of human intelligence, many important insights into how human brains work have emerged. Important properties of cognitive architectures are associated with representation, organization, utilization, and acquisition of knowledge. They usually include a programming language that lets one construct knowledge-based systems of various kinds.

SOAR (State, operator, and result; [LNR87]) has its root in the classical artificial intelligence [New90] and is one of the first cognitive architectures introduced (it is continuously being developed since the early 1980s). SOAR incorporates a wide range of problem solving methods and learns all aspects of the tasks to perform them. The long-term memory in SOAR is represented by production rules. These rules

are organized in terms of operators associated with problem spaces. The working memory in SOAR cognitive architecture contains all the knowledge that is relevant to the current situation. Tasks are formulated in terms of goals to be achieved. Typical processing cycle repeatedly suggests, selects, applies, and terminates operators of the current to the next problem state, making one decision at the time. When one goal is not achievable, SOAR creates a new goal and selects a new operator. SOAR is capable of generating a goal hierarchy, by decomposing problems into subproblems. When a new result is produced, the system learns one or more *chunks* and stores them as production rules. A production rule triggers in a new situation that is similar along relevant dimensions.

ACT-R (Adaptive control of thought-rational; [ABB⁺04]) is notably different from SOAR by its strong emphasis of producing a psychologically motivated cognitive model. It uses empirical data derived from experiments in cognitive psychology and brain imaging. ACT supports two different long-term memories: declarative memory and procedural memory. A declarative memory contains knowledge about facts and events in a form of the so-called *chunks*, while knowledge in procedural memory is available in terms of production rules. Each declarative chunk is associated with internally stored information about its past usage, while production rules have associated expected costs (in terms of time or number of steps to achieve the goal) and probability of success.

ICARUS [LC06] encodes knowledge as reactive skills, each of which specifies the goal-relevant reactions to a class of situations. The key processes in ICARUS include inference mechanism, goal selection, skill execution, problem solving, and learning. ICARUS also focuses on only one goal at a time. Whenever an applicable skill to achieve its current goal could not be found, it employs means-ends analysis as its problem solving strategy, which involves the decomposition of a problem into subgoals. After accomplishing a subgoal, the agent returns to the parent goal. Skill learning in ICARUS occurs as a result of problem solving activities. Namely, a skill is learnt when an agent is able to execute some action successfully.

Langley *et al.* [LLR09] advocate that despite the many advances that have occurred over several decades of research, cognitive architectures still have many limitations and open issues such as (1) relatively poor categorization and understanding, (2) limited possibilities of encoding knowledge in different yet interrelated formalisms, (3) limited range of knowledge utilization strategies, and (4) lack of robust and flexible learning mechanisms for extended operation in unfamiliar and/or more

complex domains.

Anderson [And93a] advocates that two key features often observed of human problem solving are *difference reduction* and *subgoaling*. Problem solvers tend to choose actions that approach them to the goal state and are even reluctant to pursue paths that temporarily take them into the opposite direction. Anderson and Kushmerick [AK90] demonstrated that the time to make a move in the Tower of Hanoi task is strongly correlated with the number of subgoals necessary to complete the particular move. These findings speak in favor of means-ends analysis, which is so often used in various cognitive architecture systems.

While the theories of human cognition and integrated cognitive architectures shed light on the way human approach to problem solving and on the possibilities of using computer programs to model human problem solving, effective formalized ways of measuring human problem-solving performance remain to be discovered.

A second question that should be addressed with respect to human problem solving is: How difficult is the problem to solve? Campbell [Cam88] reviewed task complexity in light of the following aspects: (1) primarily a psychological experience, (2) an interaction between task and the persons' characteristics, and (3) a function of objective task characteristics. The latter aspect is obviously the only one that could be tackled by the methods of artificial intelligence alone. Effective ways to formalize difficulty for a human of a given task or a set of given tasks also remain undiscovered.

1.5.2 Artificial Intelligence in Education

The field of artificial intelligence and education is grounded in three academic disciplines: computer science, psychology, and education, which all contribute to the development of the interdisciplinary field of Intelligent Tutoring Systems (ITS) [Woo08]. Research on intelligent tutoring serves two goals. Beside the obvious goal of developing systems for automating education, an equally important goal is to explore epistemological issues concerning the nature of the knowledge that is being tutored and how that knowledge can be learned [ABCL90].

The idea of using computers to enhance learning began with the Computer-Aided (or Assisted) Instruction (CAI) systems (*e.g.*, see [KBW83]), where computers were used mostly for presentation of student material. The main deficiency attributed to these systems is their static behavior; they are unable to interact with students or adjust to the specific student needs. Simple computer assisted instruction systems

suffer from the fact that in general they do not know the subject matter they are teaching. Intelligent tutoring systems use artificial intelligence (AI) formalisms to represent knowledge in order to improve on CAI systems [Yaz86].

One-to-one tutoring with personal human tutors provide a highly efficient learning environment and have been estimated to increase mean achievement outcomes by as much as two standard deviations [Blo84]. Education based on CAI systems has also been well documented to improve learning at the elementary, secondary, higher-, and adult-education levels. A meta-analysis of several hundred well-controlled studies showed that student scores increased by 10% to 20%, the time to achieve goals decreased by one-third, and class performance improved by about one-half standard deviation. The current state-of-the-art intelligent tutoring systems are estimated to increase mean achievement outcomes by about one standard deviation [Woo08].

To our knowledge, intelligent tutoring systems that have been most successful at aiding student learning are Model-Tracing Tutors (MTT) [AP91] that are commonly used in teaching problem solving domains and allow the tutor to follow the problem-solving steps of the student through the use of a detailed cognitive model of the domain. MTTs have had considerable success in improving student learning [ACKP95]. Carnegie Learning, a company founded by researchers from Carnegie Mellon, produced the commercial version of such tutor for use in high school mathematics classes, which was used in about 10% of the U.S. high school math classes in 2007 [Woo08]. A second successful and widely used model-tracing tutor is the Andes Physics Tutor [VLS⁺05].

The core of model-tracing tutoring systems is an expert module that contains the cognitive model of the domain. Such cognitive models are usually based on a theory of human cognition, for example, the Carnegie Learning tutors are based on ACT-R, a learning theory and cognitive architecture framework [And93b]. ACT-R assumes that skill knowledge is initially encoded in a declarative form when students read or listen to a lecture. Students employ general problem-solving rules to apply declarative knowledge (concepts, facts, procedures etc.), but with practice, domain-specific procedural knowledge is formed. ACT-R assumes that procedural knowledge can be represented as a set of independent production rules that associate problem states and problem-solving goals with actions and consequent state changes.

Several research issues limit the use of Model-Tracing Tutors. Production rules have limited generality, and all model-tracing tutors suffer from the difficulty of acquiring problem-solving models, which requires cognitive task analysis, an enormous

undertaking for any nontrivial domain. Cognitive analysis is typically performed manually and is tedious, time consuming, and error prone. Student models are often hand-coded and remain fossilized unless extended with human help. Additionally, this method is nearly impossible to reproduce for disciplines in which no well-developed psychological theory exists, such as medical diagnosis or law [Woo08].

Thus, whereas intelligent tutoring are proving to be useful they are also difficult and expensive to build [Mur99], mainly because building the expert model is difficult. The MTT implicitly require complete domain knowledge, which requires a lot of knowledge engineering. And although many authoring tools (tools for building an ITS without the need of a programmer) were proposed, they have not been shown to be usable for modeling domain expertise [Mur99].

Building the expert module of a tutoring systems is similar to building the knowledge base of an expert system, where machine learning is commonly used as an alternative way of obtaining the expert knowledge [FR86]. However, it should be stressed that the kind of knowledge required for an ITS (including the domain knowledge component) is different to that required for an expert system in the domain [Cla87]. It is quite possible to have an expert system that can perform the task well but that is poor at teaching or even explaining its reasoning because so much knowledge remains implicit [Twi92]. While it was shown that machine learning can be successful in building knowledge bases for expert systems [LS95] in terms of performance, the major problem with this approach is that these models usually do not mimic the cognitive processes (how an expert or a student solves problems in the given domain), which is the most important requirement of the expert module. Sison and Shimura [SS98] shed light on the difficulties of using machine learning for automating the construction of student models as well as of the background knowledge necessary for student modeling.

In the construction of intelligent tutoring systems, the acquisition of background knowledge, either for the specification of the teaching strategy, or for the construction of the student model, identifying the deviations of students' behavior, remains one of the unsolved problems [Ant08]. A still unanswered question is: is it possible to conceptualize (semi)automatically the domain in a way that conforms to the way the experts want to have their knowledge organized and presented?

1.6 Research Summary

The thesis is divided into three parts. In the first part (Chapters 2, 3, and 4), “Search and Knowledge for Estimating Human Problem Solving Performance,” we address the following question.

- How can we develop methods based on computer heuristic search for evaluating human problem solving performance?

This includes answering the following two research questions (RQs).

RQ₁ How can a computer be used to assess a human’s problem-solving performance?

RQ₂ How can a machine problem solving model be used to assess the difficulty of a set of given problems for a human?

In the second part (Chapters 5, 6, and 7), “Search and Knowledge for Improving Human Problem Solving Performance,” we address the following question.

- How can we develop methods based on computer heuristic search for improving human problem solving performance?

This includes answering the following two research questions.

RQ₃ How can machine problem solving be used in tutoring, for teaching a human to solve problems in a given problem domain?

RQ₄ How can knowledge represented in a form suitable for the computer, be transformed into a form that can be understood and used by a human?

The third part (Chapters 8 and 9), “On The Nature of Heuristic Search in Computer Game Playing,” aims at improving the understanding of properties of heuristic search and consequences of the interaction between search and knowledge that typically occurs in both human and machine problem solving, in particular with respect to computer game playing.

1.7 Thesis Overview

The thesis is organized as follows.

In Chapter 2 we devise a comparison of World Chess Champions, as a case study of estimating problem-solving performance using a heuristic-search based program. It is based on the evaluation of the games played by the World Chess Champions in their championship matches. We were interested in the chess players' quality of play regardless of the game score, performing computer analyses of individual moves made by each player. Since the champions are top-level human experts in the game of chess, evaluating their performance is a great challenge. The evaluation was performed by the chess-playing program CRAFTY.

Chapter 3 provides a thorough analysis of credibility of our method of estimating problem-solving performance in the game of chess. We repeated our computer analysis of World Chess Champions, using various chess programs at different levels of search. We provided experimental results and theoretical explanations to show that, in order to obtain a sensible ranking of the chess players using our method, it is not necessary to use a computer that is stronger than the chess-players themselves.

Chapter 4 addresses another important aspect of problem solving: How difficult are the problems to solve? From the chess player's point of view, our basic method for estimating problem-solving performance is particularly crude in that it does not take into account the differences in the average difficulty of the positions the players were faced with. We devise a heuristic-search based method to assess average difficulty of positions used for estimating the champions' performance and present the results of applying this method in order to compare chess players of different playing styles.

In Chapter 5, we introduce some elements of an intelligent computer system aimed to provide commentary in a comprehensible, user-friendly and instructive way. The main idea is to use a heuristic-search program to provide results of heuristic search and heuristic-evaluation function's features to describe the changes between the root node and the goal node. The features can then be combined to form higher-level concepts understandable to humans. Additional knowledge could be introduced into the system to describe the differences between the root node and the goal node. The descriptions can then be used both for the purpose of intelligent tutoring by providing knowledge-based feedback to students, and for the purpose of annotating. Chess is again used as a research domain, however, the scope of our research findings are intended not to be limited to chess only. The most natural way of annotating

chess games in chess literature (for example, [Kas06]) is the following: commentators usually support suggested moves with variations, and (1) comment on changes that would occur and/or (2) describe the envisioned position at the end of the variation. Our approach follows this natural way of providing commentary. Chess has been used several times as a domain in scientific research for automatic annotating ([GGM93], [Sei94], and [HB96]), however, the demonstrated concepts all have a common weakness - the inability to practically extend annotations to the entire game of chess. Our approach is not to be bound to such limitations.

In Chapter 6, we demonstrate a new approach, based on argument-based machine learning [MvB07], to the formalization of complex patterns for the purpose of commenting and/or teaching. As knowledge is one of the key components of every intelligent computer system, the process of obtaining knowledge from domain expert, called knowledge elicitation, is known to be a difficult task and thus a major bottleneck in building a knowledge base in expert systems [Fei03]. Components of a heuristic-search program's evaluation function alone are hardly sufficient for making in-depth comments, thus obtaining knowledge for construction of more complex positional features is desirable for successful annotating software. Defining complex patterns requires powerful knowledge-elicitation methods. In the presented case study, we considered the elicitation of the well-known chess concept of the bad bishop and showed that argument-based machine learning enables such a method.

In Chapter 7, we demonstrate how to synthesize semi-automatically knowledge usable for intelligent tutoring purposes from databases that contain perfect information. Complete tablebases [Tho82], indicating best moves for every position, that exist for chess endgames, served as a source of such perfect information. We developed a novel approach to deriving meaningful concepts and strategies usable for constructing a heuristic evaluation function for commenting and/or teaching purposes. Our approach combines ideas from argument-based machine learning with specialized minimax search to extract a strategy for solving problems that require search. We also explain the guidelines for an interaction between the machine and the expert in order to obtain textbook instructions suitable for teaching how to deliver checkmate in a difficult chess endgame, and how these instructions, including illustrative diagrams, could be derived semi-automatically from such a model.

In Chapter 8, we analyze the properties of successful evaluation functions in game playing. The *monotonicity property* of heuristic evaluation functions for games is introduced. That is a property of successful heuristic evaluation functions for games.

Namely, that backed-up values of the nodes in the search space have to tend to monotonically approach to the terminal values of the problem state space with the depth of search. The experimental results show that the evaluation functions used in typical chess programs tend to have this property that enables the program to play with a sense of direction towards a desirable goal. We demonstrate that backed-up heuristic values therefore do not approximate some unknown “true” or “ideal” heuristic values with increasing depth of search, in contrast to what is generally assumed in the literature, and point out that heuristic evaluation functions should *not* respect the minimax relation. That is, backed-up heuristic evaluation values should not be invariant along the game tree, as game-theoretical values in the theoretical minimax model are. We also argue that heuristic evaluations obtained by a search to different search depths are *not* directly comparable, and demonstrate that the same backed-up heuristic values obtained by different evaluation functions do *not* necessarily reflect the probability of winning in the same way.

In Chapter 9, we study experimentally additional factors which influence the behavior of diminishing returns with increased search. Deep-search behavior and the phenomenon of diminishing returns for additional search effort have been studied by several researchers, whereby different results were obtained on the different data sets used. Our results were obtained on a large set of more than 40,000 positions from real chess games using chess programs CRAFTY, RYBKA, and SHREDDER. We demonstrate that the rate of changed decisions that arise from search to different depths depends on (1) the quality of knowledge in evaluation functions, (2) the value of a node in the search space, and to some extent also on (3) the phase of the game.

1.8 Contributions to Science

The thesis is a contribution to computer science and artificial intelligence. The research done in the scope of the thesis was performed in the framework of human and computer game playing, and the game of chess was used as the experimental domain. Many researchers used the game-playing platform and the domain of chess in their experiments, the explicit or implicit message of their works being that the results for chess are generalizable to other domains. Although we did not aim at providing specific evidence for that, we believe that the below stated contributions to science also have a potential of being extendable to several other games, as well as to some other domains where heuristic search is applicable.

The main contributions of the thesis are as follows.

1. A new method, based on computer heuristic search, for the assessment of *human problem solver's performance*, and an analysis of appropriateness of this method. [Chapters 2 and 3]
2. A new method for the assessment of the *difficulty*, for a human, of a given set of problems, based on the computer's solving of these problems. [Chapter 4]
3. A novel approach, based on computer heuristic search, to automated generation of human understandable commenting of decisions in problem solving. [Chapter 5]
4. A novel approach to the formalization of complex patterns for the purpose of annotating problem solving decisions and/or intelligent tutoring. [Chapter 6]
5. A novel approach to semi-automatic synthesis of human-understandable knowledge suitable for teaching how to solve problems in a given problem domain. [Chapter 7]
6. An extensive analysis of the monotonicity property of successful heuristic evaluation functions for games. Namely, that backed-up values of the nodes in the search space have to tend to monotonically approach to the terminal values of the problem state space with the depth of search. This means that backed-up heuristic values therefore do not approximate some unknown "true" or "ideal" heuristic values with increasing depth of search, in contrast to what is generally assumed in the literature, and that successful heuristic evaluation functions should *not* respect the minimax relation. That is, backed-up heuristic evaluation values should not be invariant along the game tree, as game-theoretical values in the theoretical minimax model are. [Chapter 8]
7. An empirical proof of a novel finding that the rate of changed decisions that arise from search to different depths depends on:
 - the quality of knowledge in evaluation functions, and
 - the true value (relative to a fixed search depth) of a node in the search space. [Chapter 9]

Part I

Search and Knowledge for Estimating Human Problem Solving Performance

Chapter 2

Computer Analysis of World Chess Champions

This chapter is an updated and abridged version of the following publication:

1. Guid, M. and Bratko, I. Computer Analysis of World Chess Champions. *ICGA Journal*, Vol. 29, No. 2, pp. 65-73, 2006. [GB06]

It also includes some materials from the following publication:

1. Guid, M., Pérez, A., and Bratko, I. How Trustworthy is CRAFTY's Analysis of World Chess Champions? *ICGA Journal*, Vol. 31, No. 3, pp. 131-144, 2008. [GPB08]

In this chapter, we introduce a method, based on computer heuristic search, for evaluating problem-solving performances of World Chess Champions, top-level human experts in the game of chess.

Establishing heuristic-search based computer programs as an appropriate tool for estimating problem-solving performance in chess may seem impossible, since it is well known that both programs' evaluations and programs' decisions tend to change as depth of search increases. It is very likely that computer chess programs will continue to change their decisions with searching deeper in heuristic-search based computer analyzes for many years to come, and that the frequencies of changes will remain significant for all feasible search depths. Not to mention that the cease of decision changes at some particular search depth does not guarantee optimal play at all. Also, it is even unclear what "optimal" is? For a human, it is not necessarily

the shortest path to win, but a kind of “easiest” and most reliable one, that is one that minimizes the risk – the probability for a human to make a mistake. It is, however, not feasible to determine such probabilities. All these issues seem like a giant obstacle on the way to establishing heuristic-search based methods as competent problem-solving performance estimators, especially in complex games like chess.

Nevertheless, we will demonstrate that heuristic search can be used reliably for the purpose of evaluating problem-solving performance. In this chapter, we present the rankings of World Chess Champions based on the scores obtained at the highest practically feasible search depth. This may seem as the only reasonable alternative, since it is well known that the programs’ chess strength increases with depth of search. In Chapter 3, however, we will show that the rankings based on the obtained scores are surprisingly stable over a large interval of search depths, at least for the players whose score significantly deviates from the others.

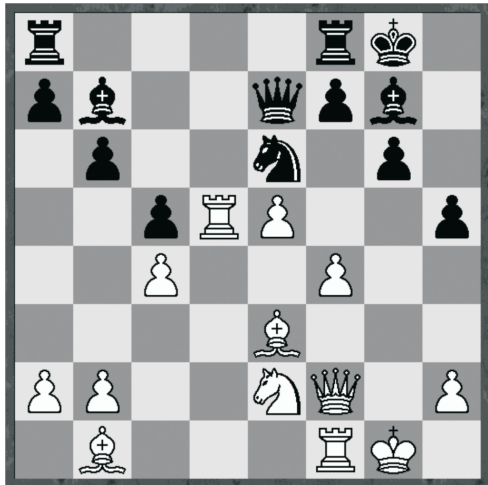
2.1 Can Heuristic Search be Useful for Estimating Problem Solving Performance?

There are many arguments in favor of heuristic-search based computer programs being an appropriate tool for estimating problem-solving performance. In contrast to humans, they:

1. have an enormous computing power,
2. use numerical values as evaluations,
3. adhere to the same rules all the time,
4. are not influenced by emotions.

Computer programs therefore have a capability of being more consistent than human observers, and can deal with incomparably more observations in a limited time. In chess, they are particularly good at evaluating tactical positions, where a great deal of computation is required.

As the basis for estimating each player’s performance we chose the average differences between heuristic evaluations of the players’ decisions and heuristic evaluations of computer’s decisions, both obtained after a fixed depth of search. This may



Depth	Best move	Evaluation
2	Bxd5	-1.46
3	Bxd5	-1.44
4	Bxd5	-0.75
5	Bxd5	-1.00
6	Bxd5	-0.60
7	Bxd5	-0.76
8	Rad8	-0.26
9	Bxd5	-0.48
10	Rfe8	-0.14
11	Bxd5	-0.35
12	Nc7	-0.07

Figure 2.1: Botvinnik-Tal, World Chess Championship match (game 17, position after white's 23rd move), Moscow 1961. In the diagram position, Tal played 23...Nc7 and later won the game. The table on the right shows CRAFTY's decisions and evaluations as results of different depths of search. As it is usual for chess programs, the decisions and the evaluations vary considerably with depth. Based on this observation, a straightforward intuition suggests us that by searching to different depths, different rankings of the problem solvers, in this case chess players, would have been obtained, had their performance been estimated using a heuristic-search based program. However, as we will demonstrate, the intuition may be misguided in this case.

appear surprising, since different search depths may result in large differences in position evaluations and in completely different choices (see Figure 2.1). However, in sufficiently large samples of data that is a subject of computer analysis, larger errors of the program (in terms of unfair differences in assigned evaluations) are expected to cancel out through statistical averaging. Our goal is no more and no less than to devise a method for estimating the problem-solving performance as best s possible. It would lead to the the correct ranking of problem solvers' performances. Having this point in mind, we expect that the computer estimator of the problem-solving performance is trustworthy already when it is equally unfair to all problem solvers.

2.2 Determining Best Chess Player in History

Who is the best chess player of all time? This is a frequently posed and interesting question, to which there is no well founded, objective answer, because it requires a comparison between chess players of different eras who never met across the board.

With the emergence of high-quality chess programs a possibility of such an objective comparison arises. However, so far computers were mostly used as a tool for statistical analysis of the players' results. Such statistical analyzes often do neither reflect the true strengths of the players, nor do they reflect their quality of play. It is common that chess players play against opponents of different strengths and it is well known that the quality of play changed in time. Furthermore, in chess a single bad move can decisively influence the final outcome of a game, even if all the rest of the moves are excellent. Therefore, the same result can be achieved through play of completely different quality.

The most complete and resounding attempt made to determine the best chess player in history has been put forward by Jeff Sonas, who has become a leading authority in the field of statistical analysis in chess during the past years. Sonas [Son] devised a specialized rating scheme, based on tournament results from 1840 to the present. The rating is calculated for each month separately, with the player's activity taken into account. A player's rating, therefore, starts declining when he is no longer active, which differs from the classic FIDE rating. Having a unified system of calculating ratings represents an interesting solution to determining a "common denominator" for all chess players. However, it does not take into account that the quality of play has risen drastically in the recent decades. The first official World Champion, Steinitz, achieved his best Sonas rating, which is on a par with ratings of recent champions, in April 1876. His rating is determined from his successes in tournaments in a time when the general quality of play was well below that of today. The ratings in general reflect the players' success in competition, but not directly their quality of play. With other words, ratings do not necessarily reflect problem-solving performance.

Other estimates about who was the strongest chess player of all times, are primarily based on the analyzes of their games as done by chess grandmasters; obviously these are often subjective. In his unfinished set of books *My Great Predecessors*, Gary Kasparov [Kas06], the thirteenth World Chess Champion, analyzes in detail numerous games of the best chess players in history and will most probably express his opinion regarding who was the best chess player ever. But it will be merely an *opinion*, although very appreciated in the chess world.

Our approach was different. We were interested in the chess players' *quality of play* regardless of the game score, which we evaluated with the help of computer analyzes of individual *moves* made by each player.

2.3 Obtaining Data for the Analysis

We evaluated fourteen classic-version World Champions, from the first World Chess Championship in 1886 to the World Chess Championship match between Kramnik and Topalov in 2006. Matches for the title of “World Chess Champion”, in which players contended for or were defending the title, were selected for analysis. Only the games with slow time control were analyzed. World Champions represent problem solvers, and chess positions they were confronted with represent problem-solving tasks.

Evaluation of each game started on the 12th move, without the use of an openings library, of course. This decision was based on the following careful deliberation. Not only today’s chess programs poorly evaluate positions in the first phase of a game, but also analyzing games from the start would most likely favor more recent champions, due to the vast progress made in the theory of chess openings. In contrast, starting the analyzes on a later move would discard too much information.

Each position was iteratively searched to fixed depths ranging from 2 to 12 ply. Search to depth d here means d -ply search extended with quiescence search to ensure stable static evaluations. Conducting search to variable depths, for example, by fixing the amount of time dedicated to search in an individual position, would likely lead to false conclusions, due to the monotonicity property of heuristic evaluation functions, as it was explained in Subsection 8.5.1. With such an approach we also achieved the following advantages.

1. Complex positions, which require processing bigger search trees to obtain an evaluation at the search depth specified, automatically receive more computation time.
2. The program could be run on different computers and still the same evaluations for a given set of positions on each of the computers are obtained.

As the second advantage suggests, searching to fixed depths assures the repeatability of the analysis. It also enabled us to speed up the calculation process considerably by distributing the computation among a network of machines, and as a consequence, searching to a greater depth was possible.

We chose to limit the search depth to 12 plies plus quiescence search. There were some speculations that a program searching 12 plies would be able to achieve a rating

that is greater than that of the World Champion [HACN90], although this speculation was based on observations of gains in strength of chess programs with each ply at shallowest search depths, and it is not likely to be correct, due to diminishing returns for searching more deeply (see Section 9.1). However, the search depth mentioned was chosen as the best alternative, since deeper search would mean a vast amount of additional computation time*.

The chess program CRAFTY was used as an estimator of problem-solving performance. We slightly modified CRAFTY for the purpose of our analysis. We changed the ‘King’s safety asymmetry’ parameter thus achieving a shift from CRAFTY’s usual defensive stance to a more neutral one where it was neither defensive nor offensive.

With each evaluated move, data was collected for search depths ranging from 2 to 12 ply, comprising:

1. the best evaluated move and the evaluation itself,
2. the second-best evaluated move and its evaluation,
3. the move made by the human and its evaluation.

We also collected data about a material state in positions from the first move on.

2.4 Three Criteria for Estimating Performance

2.4.1 Basic Criterion

The basic criterion was the average difference between the evaluations of the moves that were played by the players and evaluations of best moves suggested by computer, both obtained at a particular depth of search. These differences are referred to as players’ *scores*. The score of player P at search depth d is defined as

$$score(P) = \frac{\sum |E_{BEST(d)} - E_{PLAYED(d)}|}{N_P(d)}, \quad (2.1)$$

where $E_{BEST(d)}$ is the evaluation of the move that CRAFTY suggests as best at depth d , $E_{PLAYED(d)}$ is CRAFTY’s evaluation of the player’s move at depth d , and $N_P(d)$

*More than ten full days of computation time on 36 stand-alone computers with an average speed of 2.5 GHz were required to perform the analyzes of all games

is the number of moves analyzed for player P at particular depth. The sum is over all the moves analyzed for the player P . Based on the players' scores, rankings of the players are obtained in such way that a lower score results in a better ranking.

The following limitation was imposed upon this criterion. Moves, where both the move made and the move suggested had an evaluation outside the interval $[-2, 2]$ at the highest search depth, were discarded and not taken into account in the calculations of the scores. The reason for this is the fact that a player with a decisive advantage often chooses not to play the best move, but rather plays a move which is still "good enough" to lead to victory and is less risky. A similar situation arises when a player considers his position to be lost – a deliberate objectively worse move may be made in such a case, to give the player a higher practical chance to save the game against a fallible opponent. Such moves are, from a practical viewpoint, justified. Taking them into account would wrongly penalize players that used this legitimate approach trying (and sometimes succeeding) to obtain a desired result. All positions with evaluations outside the interval specified were declared lost or won.

The basic criterion can be criticized on the basis that it does not take into account the differences in the average difficulty of the positions played by different players. Nor does this score-based criterion take into account another important aspect, that is the differences between the playing styles of different players. Moreover, if SSDF ratings ([Kar08], see Section 9.4) are comparable with official FIDE ratings, then CRAFTY's chess rating is lower than the rating of many of the players analyzed, which makes this approach of estimating World Champions' performances based on such criterion highly counter-intuitive and seemingly highly questionable. We will return to this issues in Chapter 3 and provide a refinement and a justification for this approach.

2.4.2 Blunder Criterion

The rankings of the players according to this criterion will be devised based on the rates of blunders that occurred in their games, so that the lower the blunder rate of the player the better his ranking is.

Big mistakes or *blunders* can be quite reliably detected with a computer, to a high percentage of accuracy. Individual evaluations could be inaccurate, but such inaccuracies rarely prevent the machine from distinguishing blunders made in play from reasonable moves.

Detection of errors was similar to the basic criterion. We used a measure of difference between evaluations of moves played and evaluations of moves suggested by the machine as the best ones. We label a move as a blunder when such difference exceeds some high value, and provide results for various such “high values”. We also discarded moves where both evaluations of the move made by a player and the move suggested by the machine lie outside the $[-2, 2]$ interval, due to reason mentioned in Subsection 2.4.1. Successful blunder detection inevitably depends on chess strength, therefore only blunders detected at the highest available search depth were considered. We labeled a move as a blunder when the numerical error as seen by CRAFTY at highest available depth exceeded the value of 1.00.

There is a problem with taking a particular value at particular search depth, in terms of the difference between the played move and best evaluated move, as a big mistake or a blunder.[†] Namely, if we label a move as a blunder when the numerical error (as seen by the computer program) exceeds 1.00, then the big differences in the two evaluations such as 0.99 remain unnoticed by this measurement. Since the blunder rates of World Champions tend to be very low, it is easy to expect different rankings when the value labeled as a blunder changes only by a margin.

In order to obtain as objective results as possible, we therefore chose to vary the value of numerical error that is labeled as a big mistake or a blunder. We performed five measurements of blunder rates, the values of numerical error as seen by CRAFTY at search depth of 12 plies being labeled as a blunder being 0.8, 0.9, 1.0, 1.1, and 1.2. The final result of blunder-rate measurements was the averaged result of all five measurements performed at this (highest available) depth.

2.4.3 Best Moves Criterion

The percentage of best moves played alone does not actually describe the quality of play as much as one might expect. In certain types of position it is much easier to find a good move than in others. Experiments showed that the percentage of best moves played is correlated to the difference in evaluations of the best and second-best move in a given position. The greater the difference, the better was the percentage of player’s success in making the best move (see Figure 2.2). For each interval of the difference in evaluations of two best moves shown in the figure (the step was

[†]We also cannot know the exact interpretation of what the value of 1.00 really means - see Subsection 8.5.1

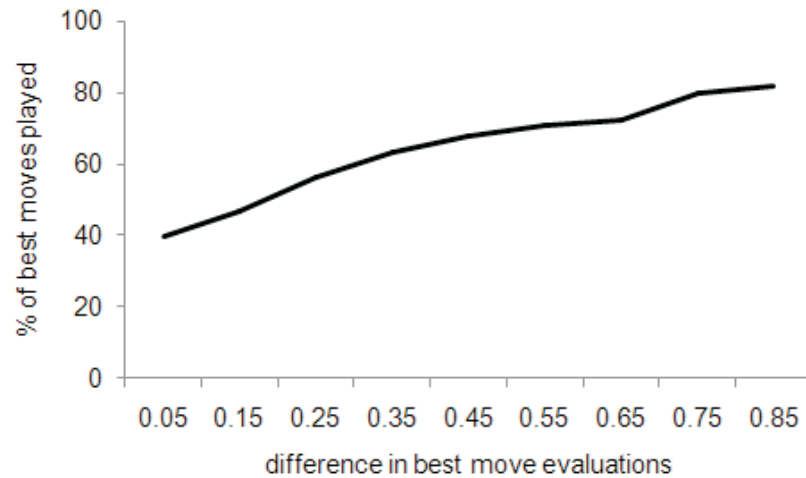


Figure 2.2: Rate of best moves played according to the computer depending on the difference in evaluations of the best and second-best move.

0.1, only intervals represented by at least 500 measurements are included), average best-move played rates are displayed.

Such a correlation makes sense, because the bigger the difference between the best two moves, the greater the error made when selecting the wrong move. The height of the curve is amplified by the fact that we are dealing with World Champions, experts at the game of chess. Analyzes of weaker players would give a curve of lesser height.

By analyzing the correlation between (1) the percentage of best moves played and (2) the difference in best two moves' evaluations, we derive information about the quality of each individual player. It turned out that curves for individual players differ significantly. This behavior served as a basis for creating the following criterion, used to infer information on the quality of individual players. For greater clarity, we will call it "the best moves criterion."

The scores by this criterion were obtained as follows. For each player we calculated the distribution of moves across separate intervals of the difference in evaluations of two best moves (where the step was 0.1). We also calculated an average distribution for all players combined. Given this average distribution, we then determined the expected percentage of the best moves played for each individual player. Due to reasons mentioned in Subsection 2.4.1, we did not count clearly lost or won positions in this statistics.

2.5 Player's Performance w.r.t. Difficulty of Positions

The main deficiency of the three criteria detailed above is in the observation that there are several types of players with specific properties, to whom the criteria do not directly apply. It is reasonable to expect that *positional players* on average commit fewer mistakes due to the somewhat less complex positions in which they find themselves as a result of their style of play, than *tactical players*. The latter, on average, deal with more complex positions, but are also better at handling them and use this advantage to achieve excellent results in competition.

From the chess player's point of view, the three criteria are thus particularly crude in that they do not take into account the differences in the average difficulty of the positions played by different players. It may seem that the third of the above criteria provides a possible solution to this problem, since it was observed that the difficulty of chess positions may be somehow related to the difference in evaluations of the best and second-best move. However, it is highly unlikely that determining the difficulty of chess position is a one-dimensional problem that could be expressed merely in terms of the differences in evaluations between possible moves in a given position.

We wanted to determine how players would perform when facing positions of equal average difficulty. In order to determine this, the following two topics were addressed.

1. The assessment of *difficulty* of positions.
2. *How* to take into account the differences between players in the average difficulty of the positions encountered in their games.

These topics, which are essential part of the present computer analysis and are also very important for estimating problem-solving performance in general, will be addressed separately in Chapter 4.

2.6 Results of a Computer Analysis

In this section, we will present the results of a computer analysis of the World Chess Champions, obtained by CRAFTY using a search depth of 12 plies.[‡]

2.6.1 Results According to the Basic Criterion

The basic criterion for estimating performance of the World Champions was the average difference between moves played and best evaluated moves by the computer. Results of the computer analysis according to this measure at the highest available search depth are given in Figure 2.3. The winner was the third World Champion, José Raúl Capablanca. As stated in Subsection 2.5, we expected *positional players* to perform better by this criterion than *tactical players*. Capablanca is widely renowned to be a pure positional player. In compliance with this observation, Steinitz, who lived in an era of tactical “romantic chess,” took clearly last place.

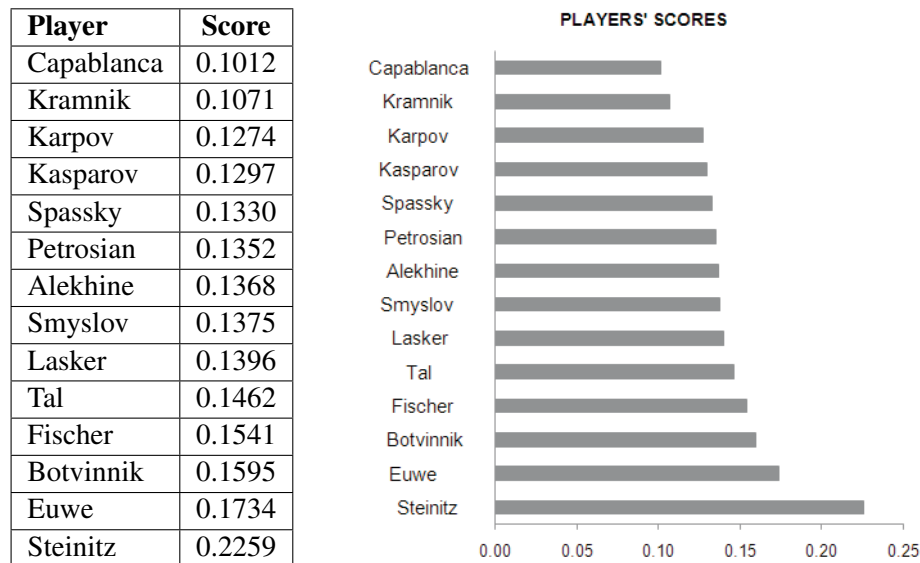


Figure 2.3: Ranking of the champions based on average differences between moves played and best-evaluated moves, according to CRAFTY at search depth of 12 plies.

[‡]Note that the results are slightly different from those presented in the paper *Computer Analysis of World Chess Champions* [GB06]. In the current text the search was limited to 12 plies, and was not extended to 13 plies in the endgames, as it was in the paper cited. For explanations why searching to a fixed depth is desirable we refer to Subsection 8.5.1.

Some of the results might appear quite surprising and may thus be considered also as an interesting contribution to the field of chess. Capablanca's outstanding score in terms of this score-based criterion will probably appear to many as such an interesting finding, although it probably should not come as a complete surprise. Gary Kasparov [Kas06] describes Capablanca by the following words: "He contrived to win the most important tournaments and matches, going undefeated for years (of all the champions he lost the fewest games)" and "his style, one of the purest, most crystal-clear in the entire history of chess, astonishes one with his logic."

As it could be seen from Figure 2.3, for majority of the players the scores are very similar. This does not appear surprising bearing in mind that the players that were a subject of the analysis were all World Champions, so the quality of play in the games at the peak of their careers may indeed not differ significantly when being compared, at least when the majority of the champions is concerned. For appropriate interpretation of the obtained scores of the players, we refer to Chapter 3.

2.6.2 Blunder-Rate Measurements

The results of the blunder-rate measurements (see Figure 2.4) show a triumph of the champions that are commonly regarded as *positional players*: Capablanca, Kramnik, Petrosian, Karpov, and Smyslov. Again, the results can be nicely interpreted by a chess expert. Kasparov [Kas06] when commenting Capablanca's games speculates that Capablanca occasionally did not even bother to calculate deep tactical variations. The Cuban simply preferred to play moves that were clear and positionally so strongly justified that calculation of variations was simply not necessary. While the results are similar to those obtained with the basic criterion (see Figure 2.3), the excellent result by Petrosian, who is widely renowned as a player who almost never blundered, cannot remain unnoticed. As before, Wilhelm Steinitz is clearly last. Of course, this ranking is even more crude for the champions that could be labeled as *tactical players*, who on average dealt with more complex positions in their games, but were also better at handling them and therefore used this advantage to achieve their best results in competition.

2.6.3 Bringing the Champions to a "Common Denominator"

Our third criterion was the expected number of best moves played providing that all players dealt with positions with equal difference between the best two moves, as it

Player	Blunders (%)
Capablanca	1.30
Kramnik	1.34
Petrosian	1.67
Karpov	1.73
Smyslov	1.80
Kasparov	2.09
Spassky	2.14
Alekhine	2.34
Lasker	2.56
Tal	2.96
Botvinnik	2.91
Fischer	3.47
Euwe	4.02
Steinitz	5.55

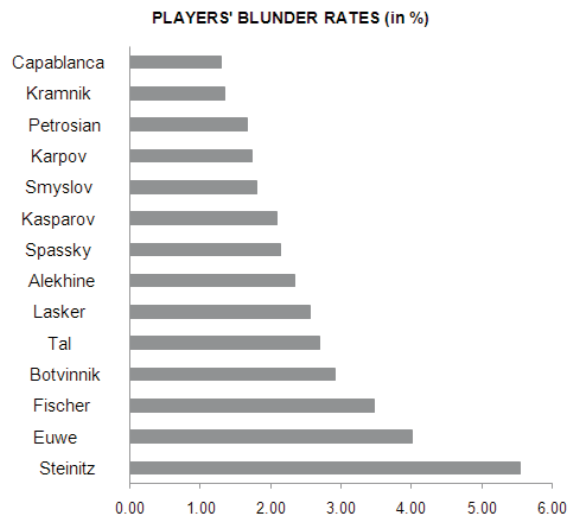


Figure 2.4: Ranking of the champions based on blunder-rate measurements according to CRAFTY at search depth of 12 plies.

was described in Subsection 2.4.3. It represents an attempt to bring the champions to a “common denominator,” by taking into account the differences in their style of play.

As it was explained in Subsection 2.4.3, the percentage of best moves played alone does not actually describe the quality of play as much as one might expect, since in certain types of position it is much easier to find a good move than in others. We demonstrated in Figure 2.2 that the percentage of best moves played is highly correlated to the difference in evaluations of the best and second-best move in a given position.

Kramnik and Alekhine, for example, had the highest percentage of best moves played (see Figure 2.5), but also the above-mentioned difference was high in the positions they were faced with (see Figure 2.6). In contrast, Capablanca, who was right next regarding the percentage of the best move played, on average dealt with the smallest difference in evaluations between the best two moves.

To make it easier for the reader, we will briefly describe again how the results of the third criterion, presented in Figure 2.7, were obtained. For each player we calculated the distribution of moves across separate intervals of the difference in evaluations of two best moves. We also calculated an average distribution for all players combined. Given this average distribution, we then determined the expected percentage of the best moves played for each individual player.

2. COMPUTER ANALYSIS OF WORLD CHESS CHAMPIONS

Player	Best (%)
Kramnik	59.26
Alekhine	56.58
Capablanca	56.30
Euwe	56.03
Fischer	55.59
Tal	54.88
Kasparov	54.68
Lasker	54.22
Karpov	54.05
Smyslov	53.22
Botvinnik	52.83
Spassky	50.87
Steinitz	50.15
Petrosian	48.32

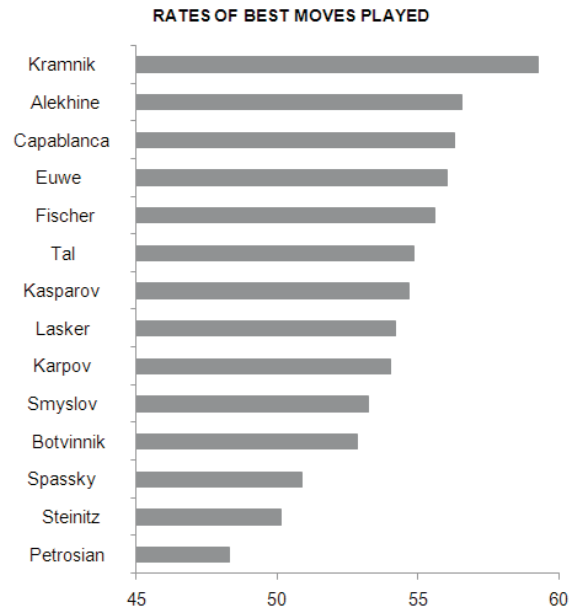


Figure 2.5: Rates of best moves played according to CRAFTY at search depth of 12 plies.

Player	Difference
Euwe	0.5952
Fischer	0.5805
Kramnik	0.5397
Alekhine	0.5233
Kasparov	0.5095
Steinitz	0.4976
Karpov	0.4906
Tal	0.4887
Spassky	0.4851
Lasker	0.4841
Botvinnik	0.4770
Smyslov	0.4236
Petrosian	0.4114
Capablanca	0.3972

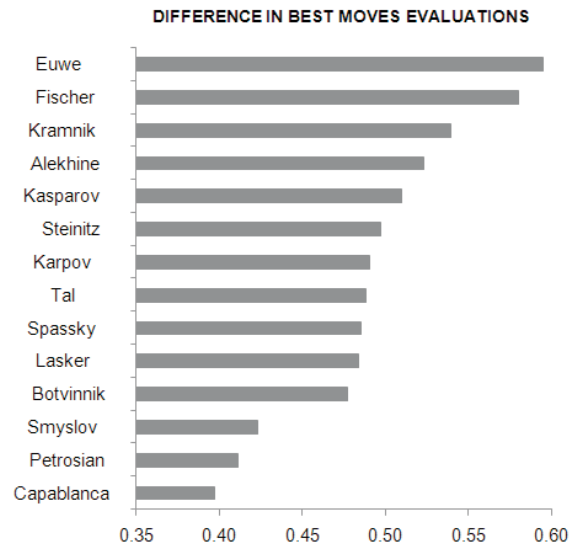


Figure 2.6: Difference in evaluations between the best two moves according to CRAFTY at search depth of 12 plies.

Player	Best (%)
Capablanca	57.08
Kramnik	56.62
Alekhine	54.67
Kasparov	53.98
Karpov	53.94
Lasker	53.50
Smyslov	53.07
Euwe	52.08
Tal	52.66
Botvinnik	52.47
Fischer	51.68
Spassky	50.80
Petrosian	49.56
Steinitz	47.50

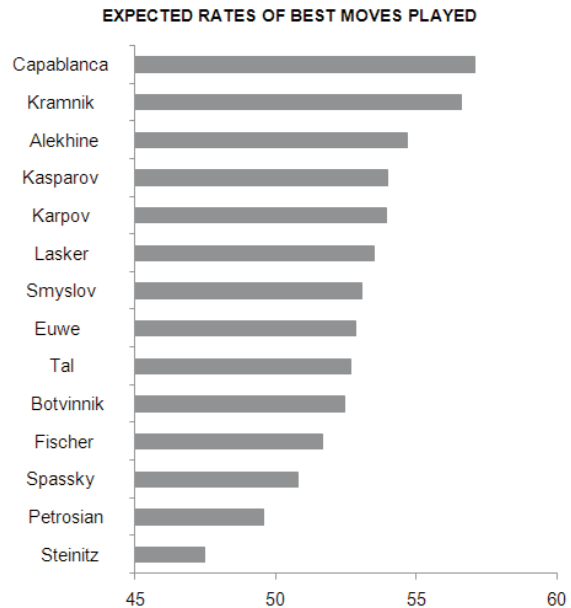


Figure 2.7: Expected percentage of the best moves played, according to the criterion described in Section 2.4.3.

The winner by the third criterion was once again Capablanca, while Steinitz again finished in the last place. The third criterion represents our first attempt to bring the champions to a “common denominator,” by taking into account the differences in their style of play. In Chapter 4, we will present the fourth criterion for estimating performance of the World Chess Champions that will also take into account the differences between players in the average difficulty of the positions encountered in their games. We will also show that the excellent result of Capablanca should be interpreted in the light of his playing style that tended towards low complexity positions.

Chapter 3

Credibility of a Heuristic-Search Based Estimator

This chapter is an updated and abridged version of the following publication:

1. Guid, M., Pérez, A., and Bratko, I. How Trustworthy is CRAFTY's Analysis of World Chess Champions? *ICGA Journal*, Vol. 31, No. 3, pp. 131-144, 2008. [GPB08]

In this chapter, we assess how reliable CRAFTY (or, by extrapolation, any other fallible chess program) is as a tool for the comparison of chess players, using the methodology presented in Chapter 2. In particular, we were interested in observing to what extent the scores and the rankings of the players are preserved at different depths of search. As it was illustrated in Figure 2.1, the decisions and the evaluations of heuristic-search based programs may vary considerably with depth of search. Nevertheless, our results show, possibly surprisingly, that at least for the players whose scores differ sufficiently from the others the ranking remains preserved, even at very shallow search depths.

We also study in this chapter how the scores and the rankings of the players would deviate if smaller subsets of positions were used for the analysis, and whether the number of positions available from World Championship matches suffices for reliable estimates of the players' deviations from the chess program.

Finally, we used three chess programs stronger than CRAFTY as estimators of problem-solving performances of the World Chess Champions.

3.1 Trustworthiness of CRAFTY's Analysis of World Chess Champions

In Chapter 2, we carried out a computer analysis of games played by World Chess Champions as an attempt at an objective assessment of one aspect of the playing strength of chess players of different times. The chess-program CRAFTY was used in the analysis. Given that CRAFTY's official chess rating is lower than the rating of many of the players analyzed, the question arises to what degree that analysis could be trusted. In this chapter, we investigate this question and other aspects of the trustworthiness of those results.

Among the three criteria considered (see Section 2.4), the basic criterion for comparison among players was the average deviation between evaluations of played moves and best-evaluated moves, both obtained as results of searching to some fixed depth. According to this criterion, Jose Raul Capablanca, achieved the best score at search depth of 12 plies, that is the highest depth that was available for our analysis (for explanation, see Section 2.3). In this chapter, we will focus mainly on the basic criterion for estimating problem-solving performance. We will provide experimental results and theoretical explanations to show that, in order to obtain a sensible ranking of the players according to the criterion considered, it is not necessary to use a computer that is stronger than the players themselves.

We will deal with the following reservations that may be imposed to the methodology used for estimating performances of World Chess Champions, using chess program CRAFTY.

1. The program used for analysis was too weak.
2. The depth of the search performed by the program was too shallow.
3. The number of analyzed positions was too low (at least for some players).

To avoid possible misinterpretation of the work presented in this chapter, it should be noted that this chapter is not concerned with the question of how appropriate this particular measure of the playing strength (deviation of player's moves from computer-preferred moves) is as a criterion for comparing chess players' ability in general. Therefore any possible interpretations of the results and rankings that appear in this chapter should be made carefully keeping this point in mind.

3.2 Variation of Rankings with Search Depth

In this section, we investigate the effects of search depth on rankings and the scores of the players, that is the average differences between player's and CRAFTY's moves. We used the same methodology as described in Section 2.3 and Subsection 2.4.1, only that now the scores (as defined in Equation 2.4.1) of the players were observed at each search depth ranging from 2 to 12.

It is well known for a long time that the strength of computer-chess programs increases with the search depth. Already in 1982, Ken Thompson compared programs that searched to different search depths. His results show that searching to only one ply deeper results in a more than 200 rating points stronger performance of the program [Tho82]. Although later it was found that the gains in the strength diminish with additional search, they are nevertheless significant at search depths up to 20 plies [Ste05]. In Chapter 9, we show that changes in decisions are also affected by the quality of knowledge in evaluation functions – the more knowledgeable evaluation function is, the less changes occur with increasing depth of search. However, even currently the strongest chess program, RYBKA 3, changes its decision on average in more than 15% cases at search depth of 12 plies (see Section 9.4). Steenhuisen conducted go-deep experiments with CRAFTY on 4,500 positions and showed that the program changes its decision in more than 10% of the cases even at search depth of 18 plies [Ste05].*

The preservation of the rankings at different search depths would therefore suggest (1) that the same rankings would have been obtained by searching deeper, and that (2) using a stronger chess program would probably not affect the results significantly, since the expected strength of CRAFTY at higher depths are already comparable with the strength of the strongest chess programs, under ordinary tournament conditions at which their ratings are measured.†

The players' scores at different search depths are presented in Figure 3.1, while

*Conducting heuristic search to such depths is very time consuming, while time spent for each additional search ply increases exponentially. In private communication, Steenhuisen acknowledged that it took several months of operating time of fourteen stand-alone computers (3.06 GHz Intel P4, 512 KByte cache, 1 GByte RAM) to analyze 4,500 positions up to 18 plies using CRAFTY.

†In the aforementioned go-deep experiments with CRAFTY Steenhuisen showed that the program changes its decision in less than 15% of the cases from search depth of 15 plies [Ste05]. According to our observations presented in Section 9.4, this result suggests that the strength of CRAFTY at 15 plies is comparable with the strongest chess program, RYBKA 3, when the latter conducts a search to a depth of 12 plies.

Figure 3.2 shows the deviations of the players' scores from the average score of all players obtained at each search depth. Some players whose rankings preserve at most of the depths are highlighted.

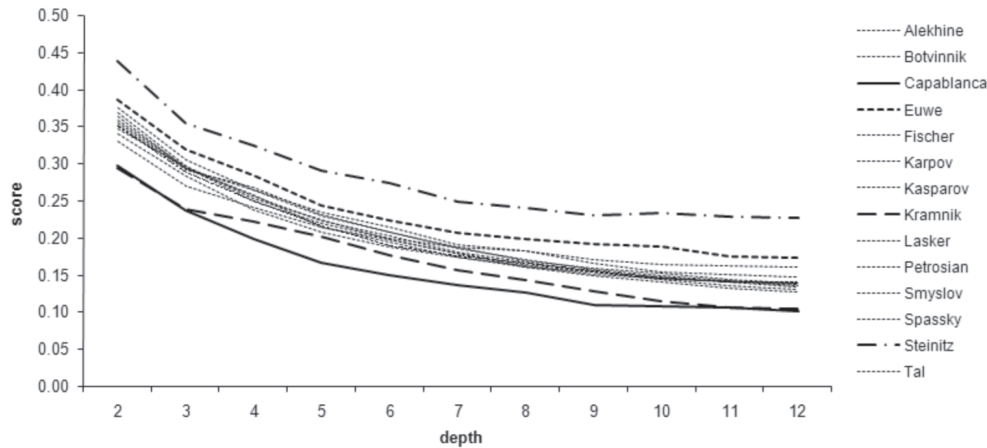


Figure 3.1: Scores of each player at different depths of search.

The results clearly demonstrate that although the scores of the players tend to decrease with increasing search depth, the rankings of the players are nevertheless preserved at least for the players whose scores differ considerably from the others. It is particularly interesting that even searching to a depth of just two or three ply (plus quiescence) does a rather good job in terms of the ranking of the players.

3.3 Robustness of Rankings w.r.t. Sample Size

The results presented in the previous section suggest that for some players the obtained rankings are preserved with depth of search. In this section we investigate the question whether the available samples of chess positions were sufficiently large to conclude that the observed differences between pairs of players are statistically significant. We then observe the stability of the obtained results, repeating the experiments on different subsets of the available positions.

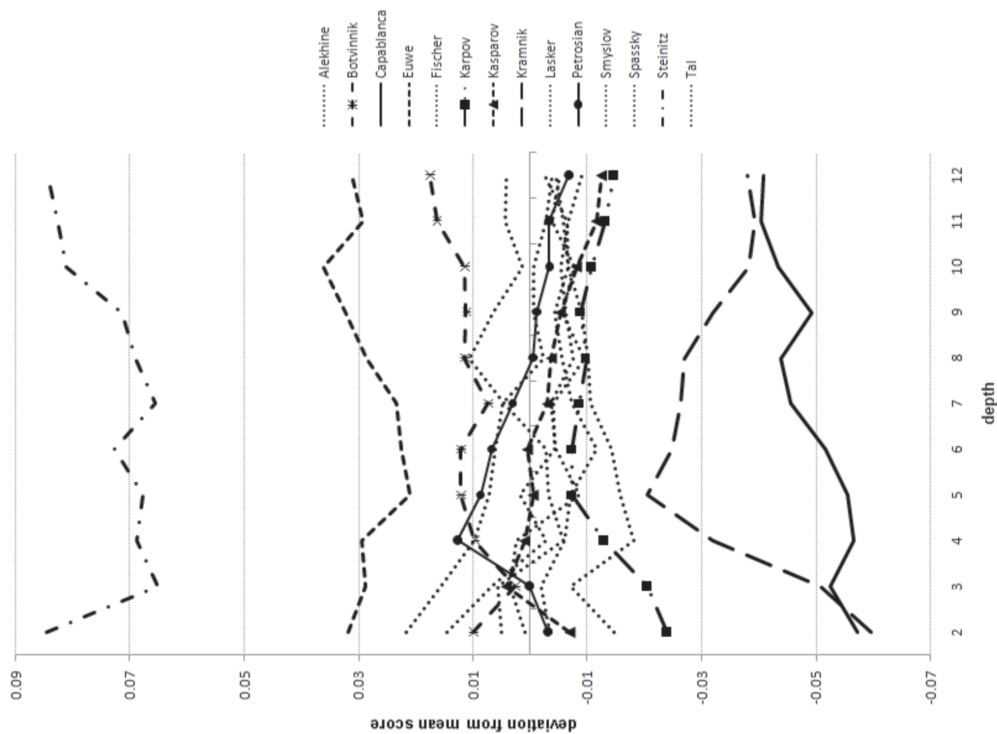


Figure 3.2: Average deviations of the players' scores from the average score of all players obtained at each depth of search. Based on the players' scores the rankings of the players were obtained. For almost all depths it holds that $\text{rank}(\text{Capablanca}) < \text{rank}(\text{Kramnik}) < \text{rank}(\text{Karpov}) < \text{rank}(\text{Kasparov}) < \text{rank}(\text{Petrosian}) < \text{rank}(\text{Botvinnik}) < \text{rank}(\text{Euwe}) < \text{rank}(\text{Steinitz})$.

3.3.1 Number of Positions for Analysis

The number of available positions varies for different players. About 600 positions only were available for Fischer, while both for Botvinnik and Karpov this number is higher than 5,000 at each depth. The exact number for each player slightly varies from depth to depth, due to the constraints of the method. Positions where both the move made and the move suggested by the computer had an evaluation (based on search to given depth) outside the interval $[-2, 2]$ were discarded at each depth.

To assess whether the set of positions available from World Chess Championship matches were sufficiently large, in order to produce reliable rankings of the players, at least for some pairs of the players, we conducted the following statistical analysis. For each player, $n=30$ samples of $m = 50$ positions were randomly chosen with

3. CREDIBILITY OF A HEURISTIC-SEARCH BASED ESTIMATOR

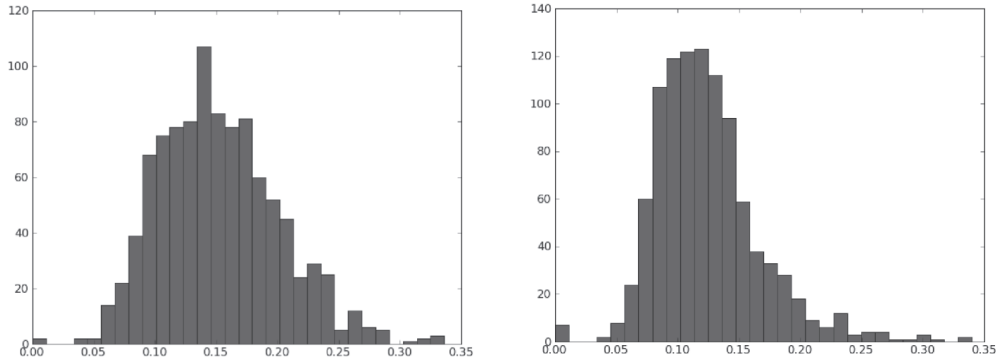


Figure 3.3: Distributions of scores in 1,000 randomly generated samples consisting of 50 positions are shown for players Fischer (left) and Karpov (right). X axis represents the player’s sample score, while Y axis represents the number of samples with the score in a given interval.

replacement from the set of all available positions for the player. For each of these positions, we observed the player’s deviations from CRAFTY’s moves (from now on we will be referring to these deviations as CRAFTY’s *differences*) previously computed for search depth of 12 plies using the method presented in Subsection 2.4.1. For each of the 30 samples, the player’s “sample score” was computed as the average of CRAFTY’s differences in the sample.

The sample scores were now used to determine statistical significance of the obtained rankings of the players. Generally speaking, for any two players P_1 and P_2 , their mutual rank $\text{rank}(P_1) < \text{rank}(P_2)$ is determined by the condition $\text{score}(P_1) < \text{score}(P_2)$. In determining statistical significance, why did we not simply use CRAFTY’s differences in the whole data set of individual positions analyzed? The reason is that the distributions of the CRAFTY’s differences on individual positions are non-symmetrical and very far from normal. Therefore we cannot apply parametric statistical tests on the original data. However, the distributions of the scores (obtained as the average CRAFTY’s differences in the sample) in samples consisted of 50 positions are approximately normal (see also the results of an experiment with 1,000 samples of 50 positions given in Fig. 3.3), so a parametric significance test as the one below is appropriate.

For a pair of players P_1 and P_2 , our null hypothesis is that their expected scores are equal. That is, if we had a very large set of positions available for each of them,

their observed scores would be indistinguishably close. The alternative hypothesis to the null hypothesis is that the players' expected scores are not equal. Now, given our limited sets of available positions, and the corresponding observed scores and their deviations for the two players, the statistical question is whether the null hypothesis can be rejected at some confidence level, say 95%. If yes, then we may with 95% confidence conclude that the expected scores of the two players are not equal, and therefore the players' performances according to this criterion are not equivalent. We use the following test (for example, see [Ros05]) to decide this. If the inequality 3.3.1 is true then the null hypothesis is rejected:

$$\sqrt{\frac{s_{\bar{X}_1}^2(m)}{n_1} + \frac{s_{\bar{X}_2}^2(m)}{n_2}} \times z > |M_1 - M_2| \quad (3.1)$$

where $s_{\bar{X}_i}^2(m)$ is the sample variance of the $n_1 = n_2 = n = 30$ scores of player P_i , \bar{X}_i is a sample score (that is an average CRAFTY's difference in a sample of m ($m = 50$) positions), and M_i is the average of the n sample scores of P_i . The value of z can be obtained from a table of the normal distribution in order to obtain the desired confidence levels. We note that a two-tailed test is appropriate for testing our null hypothesis.

We cannot apply this test in our case directly to all pairs of the players, because it is only valid for testing a single hypothesis. Since we have $\binom{14}{2} = 91$ pairs of the players, we need to test 91 hypotheses. Due to the multiple comparisons problem, a more strict test is required. One simple way to strengthen the test is to use the Bonferroni correction where the confidence level is increased by modifying $p = 0.05$ to $p = 0.05/91$, that is dividing p by the total number of tested hypotheses. The Bonferroni correction is however unnecessarily conservative. The FDR method (False Discovery Rate [BH95]), a modification of the Bonferroni correction, is a more powerful method which has become popular in many multiple hypothesis testing applications (e.g., [HvBK08]).

Before presenting the results obtained with the FDR method, we look into the following question. How large sample sizes m and n we can afford so that the statistical tests are still valid? With increased sample size, more positions are repeated in different samples, so that the samples, which ideally should be independent, may become too similar for reliably estimating the variance. Of course, the smaller the total available set is, the more repetitions we have in the samples. And again, the larger

3. CREDIBILITY OF A HEURISTIC-SEARCH BASED ESTIMATOR

the samples are, the more repetitions occur. To check the effect of increased repetitions of positions in the samples of different sizes, we split all Karpov's positions into five subsets of 1,000 positions, and measured the variance on different sized samples drawn from these subsets. We chose Karpov for this experiment because of the large set of positions available from his World Chess Championship games. Finally, we compared the obtained variances with those obtained on the whole set of more than 5,000 positions. Figure 3.4 shows the results of this experiment. The results indicate that we can afford samples of which the size may be even a rather large proportion of the total set. For example, sample size of 500 out of 1,000 seems perfectly safe. This suggests that our choice of $m = 50$ was rather conservative.

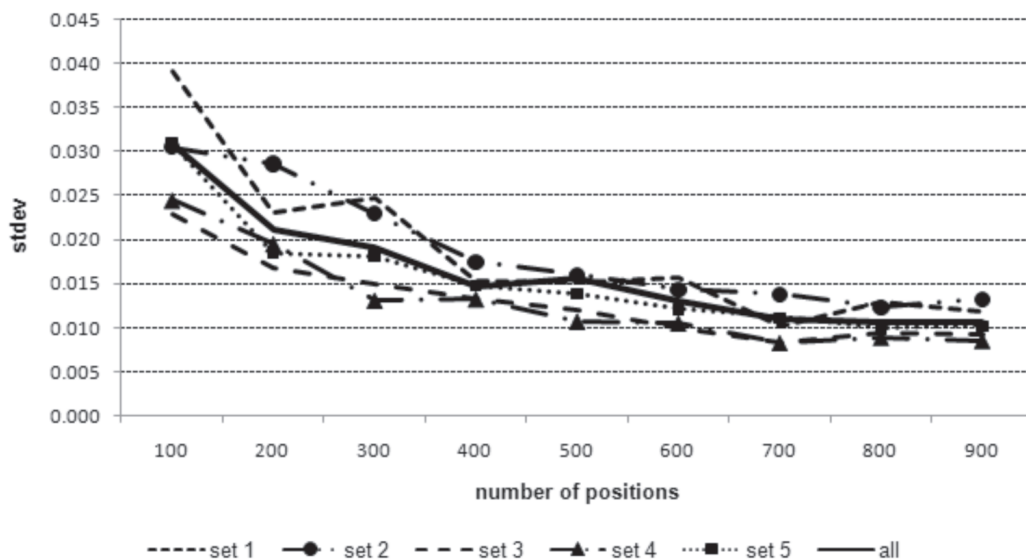


Figure 3.4: Standard deviations of the scores in 100 subsets of positions ($n = 100$) of different sizes m that were obtained on 5 data sets that each consisted of 1,000 positions from Karpov's games. Different positions were included in each data set. The results that were obtained from all available positions from Karpov's games are included as well.

The results obtained with the statistical test and by using the FDR method for multiple comparisons are shown in Table 3.1, which shows for which pairs of the players the expected scores differ at the confidence level $> 95\%$. According to the results, for 52.7% of pairs of the players we may with 95% confidence conclude that the expected scores of the two players differ significantly. Note that the sizes of the

3.3. Robustness of Rankings w.r.t. Sample Size

	Ste	Euw	Bot	Tal	Las	Fis	Smy	Ale	Pet	Spa	Kas	Kar	Kra	Cap
Capablanca	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Kramnik	X	X	X	X	X	X			X	X	X	X		
Karpov	X	X												
Kasparov	X	X												
Spassky	X	X												
Petrosian	X	X				X								
Alekhine	X	X	X			X								
Smyslov	X	X	X											
Fischer	X	X												
Lasker	X	X												
Tal	X	X	X											
Botvinnik	X													
Euwe	X													
Steinitz														

Table 3.1: The names of the players are ordered according to their scores at search depth 12 that were obtained on the whole set of positions available from their World Chess Championship matches. Pairs of the players whose expected scores differ at the confidence level $> 95\%$ are marked with 'X'. The results were obtained by the false discovery rate (FDR) procedure for multiple comparisons.

samples used in our statistical analysis were only $m = 50$ and $n = 30$. Therefore these results can be rather conservative.

The results indicate that the sets of available positions were sufficiently large to confirm reliably that for at least one half of the pairs of players their scores differ significantly. Therefore, for at least one half of the pairs of the players, their mutual rankings according to chess program CRAFTY would stay the same even if many more positions were available for the analysis.[‡] For players whose scores are very similar, however, the positions available from World Chess Championship matches do not produce statistically significant mutual rankings. This indicates that the third possible reservation stated in Section 3.1, speculating that the number of analyzed positions was too low (at least for some players), should be taken seriously, at least when looking for a firm statistical guarantee regarding the relative rankings of some pairs of the players.

[‡]Of course, this assumes that positions selected for computer analysis appropriately represent the strength of a particular player.

3.3.2 Stability of the Rankings with Search Depth

The results presented in the sequel were obtained on 100 subsets of the original data sets, generated by randomly choosing 500 positions (with replacement) from the available position samples of each player. The aim here is to study the stability of rankings across the search depths.

In order to study the stability of the scores at different samples, the standard deviation of the scores at different search depths were obtained for each of the players. The results are summarized in Figure 3.5, which shows the averages of the obtained standard deviations. The average standard deviations of the players' scores show that they are less variable at higher depths. Anyway, they could be considered practically constant at depths higher than 7. We also observed that Capablanca had the best score in 95% of all the subset-depth combinations.

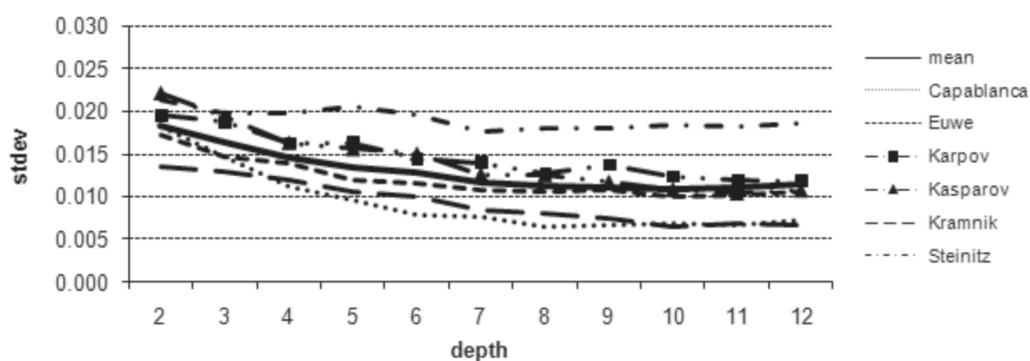


Figure 3.5: Average standard deviations of the scores of the players over 100 random subsets of 500 positions, and standard deviations of the scores of some of the players (for clarity, only a few players are included).

In order to determine the stability of the rankings (obtained in 100 subsets) across different search depths, standard deviations of the ranks of individual players at each search depth were computed. The results are summarized in Figure 3.6, which shows the average of the standard deviations of all the players. They only slightly decrease with increasing search depth and are practically equal for most of the depths.

Finally, we observed the stability of the obtained ranks for each player across different search depths, *i.e.*, how much do the players' ranks tend to change at different search depths. The results of this study are summarized in Figure 3.7, which shows

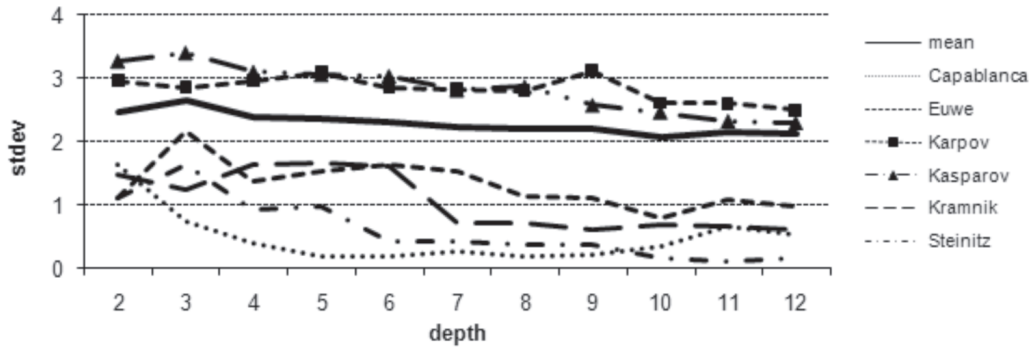


Figure 3.6: Average standard deviations of the players' ranks (obtained in 100 subsets), and standard deviations of the ranks of some of the players (for clarity, only a few players are included).

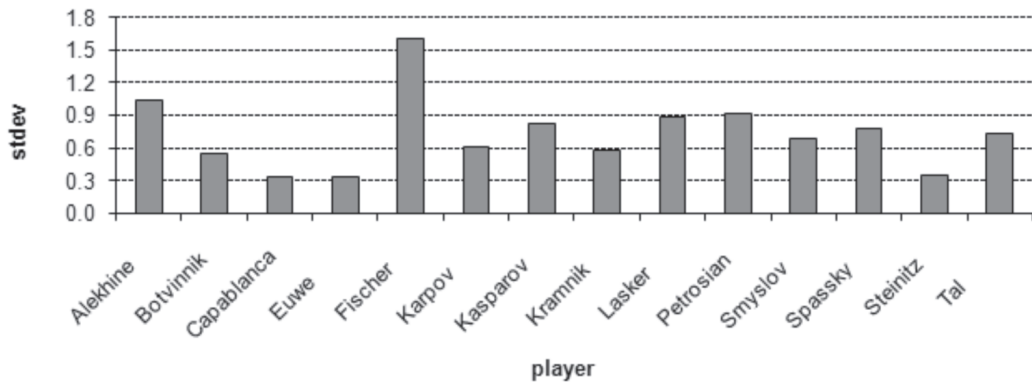


Figure 3.7: Standard deviations of the average ranks for each player across all depths.

standard deviations of the average ranks for each player across all the search depths. The low standard deviation values for most of the players (lower than 1) confirm that the rankings of most of the players on average preserve well across different depths of search.

3.4 A Simple Probabilistic Model of Ranking by an Imperfect Referee

Here, we present a simple mathematical explanation of why an imperfect evaluator may be sufficient to correctly rank the candidates. The following simple model was designed to show the two points below.

1. To obtain a sensible ranking of players, it is not necessary to use a computer that is stronger than the players themselves. There are good chances to obtain a sensible ranking even using a computer that is weaker than the players.
2. The (fallible) computer will not exhibit preference for players of similar strength to the computer.

Let there be three players and let us assume that it is agreed what is the best move in every position. Player A plays the best move in 90% of positions, player B in 80%, and player C in 70%. Assume that we do not know these percentages, so we use a computer program to estimate the players' performance. Say the program available for the analysis only plays the best move in 70% of the positions. In addition to the best move in each position, let there be 10 other moves that are inferior to the best move, but the players occasionally make mistakes and play one of these moves instead of the best move. For simplicity we take that each of the inferior moves is equally likely to be chosen by mistake by a player. Therefore player A, who plays the best move 90% of the time, will distribute the remaining 10% equally among these 10 moves, giving 1% chance to each of them. Similarly, player B will choose any of the inferior moves in 2% of the cases, etc. We also assume that mistakes by all the players, including the computer, are probabilistically independent. In what situations will the computer, in its imperfect judgement, credit a player for the "best" move? There are two possibilities.

1. The player plays the best move, and the computer also believes that this is the best move.
2. The player makes an inferior move, and the computer also confuses this same inferior move for the best.

In general in this model, the computer's estimate of a player's accuracy can be calculated as follows.

P = probability of the player making the best move

P_C = probability of the computer making the best move

P' = computer's estimate of player's P accuracy

N = number of inferior moves in a position

Then:

$$P' = P \times P_C + \frac{(1 - P) \times (1 - P_C)}{N} \quad (3.2)$$

By simple probabilistic reasoning we can now work out the computer's approximations of the players' performance based on the computer's analysis of a large number of positions. By using equation (3.2) we can determine that the computer will report the estimated percentages of correct moves as follows: player A: 63.3%, player B: 56.6%, and player C: 49.9%. These values are quite a bit off the true percentages (i.e., 90%, 80%, and 70% for players A, B, and C respectively), but they nevertheless preserve the correct ranking of the players. The example also illustrates that the computer did not particularly favor player C, although that player is of similar strength as the computer.

The simple example above does not exactly correspond to our method which also takes into account the cost of mistakes. But it helps to bring home the point that for sensible analysis we do not necessarily need computers stronger than human players.

P' is monotonically increasing with P as long as $P_C > 1 / (N+1)$. Note that P_C corresponds to random referee in the case when $P_C = 1 / (N+1)$. So according to this model, the referee only has to be better than random to obtain the ranking right, given sufficiently large samples of positions, and that the independence assumption is true. That is, the computer's choice of wrong moves is independent of the player's wrong moves. All this is not to say that a perfect referee and a referee just better than random are equally useful in determining rankings. In a realistic setting, where position sets are limited, an inferior referee is more likely to arrive at the wrong ranking because of larger statistical fluctuations in smaller samples.

3.5 A Model of Estimators of Different Strengths

Assume we have an estimator A that measures the performance of an individual M at a concrete task, by assigning this individual a score S , based on some examples of M

3. CREDIBILITY OF A HEURISTIC-SEARCH BASED ESTIMATOR

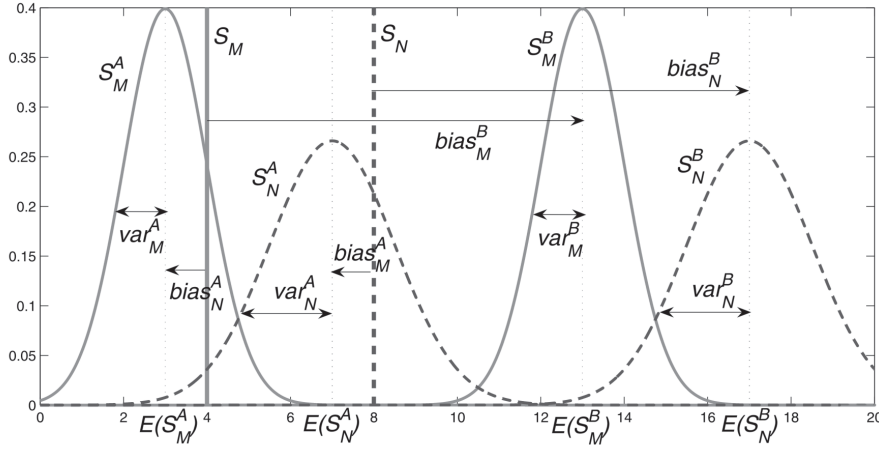


Figure 3.8: Although estimators A and B give different approximations of the true performances of individuals M and N (S_M and S_N), and A approximates the real scores more closely, since their scores are equally biased towards each individual ($Bias_M^A = Bias_N^A$ and $Bias_M^B = Bias_N^B$) and variances of the scores of both estimators are equal for each respective individual ($Var_M^A = Var_M^B$ and $Var_N^A = Var_N^B$), they are both equally suitable for mutual ranking of M and N .

performing the task. The estimator assigns different score values to the individual at different examples, and the associated variance and bias are:

$$Var_M^A = E[(S_M^A - E(S_M^A))^2] \quad (3.3)$$

$$Bias_M^A = E(S_M^A - E(S_M^A)) \quad (3.4)$$

Assuming a normal distribution of score values, the probability of an error in the relative rankings of two individuals, M and N , using the estimator A , only depends on the bias and the variance. Given two different estimators, A and B , if their scores are equally biased towards each individual ($Bias_M^A = Bias_N^A$ and $Bias_M^B = Bias_N^B$) and variances of the scores of both estimators are equal for each respective individual ($Var_M^A = Var_M^B$ and $Var_N^A = Var_N^B$), then both estimators have the same probability of committing an error (see Figure 3.8). This phenomenon is commonly known in the machine-learning community and has been frequently used, *e.g.*, in studies of performances of estimators for comparing supervised classification algorithms (for example, see [Koh95]).

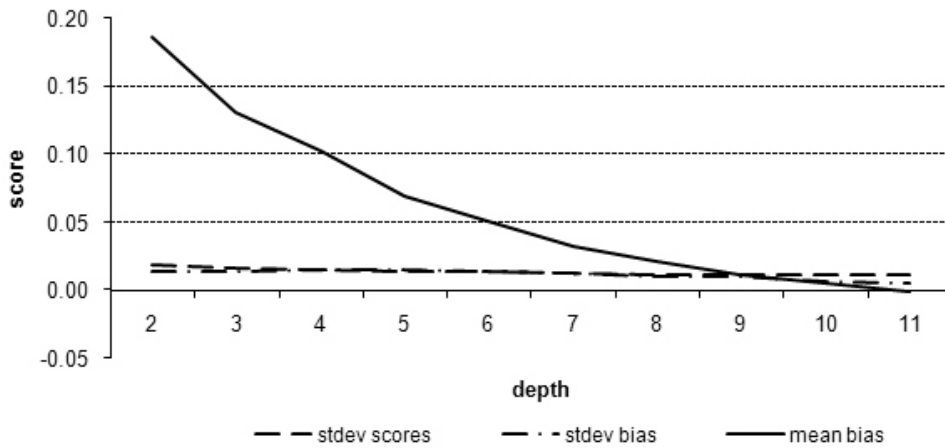


Figure 3.9: Average biases, standard deviations of them, and standard deviations of the scores with 100 subsets.

3.5.1 Variance of Players' Scores and Rankings with Search Depth

In the sequel, we analyze what happens in comparisons in the domain of chess when estimators based on CRAFTY at different search depths are used, as has been done in the work presented in this chapter.

In our study, the subscript M in S_M^A refers to a player and the superscript A to a depth of search. The true performance S_M could not be determined, but since it is commonly known that in chess the deeper search results in better heuristic evaluations (on average), for each player the score at depth 12, obtained from all available positions of each respective player, served as the best possible approximation of the true performance. The variances and the biases for each player were observed at each depth up to 11, once again using the 100 subsets of 500 positions, described in Section 3.2.

The results are presented in Figure 3.9. The standard deviation of the bias over all players is very low at each search depth, which suggests that $Bias_M^A$ is approximately equal for all the players M . The program did neither show any particular bias at any depth towards Capablanca nor towards any other player, if we assume that CRAFTY at search depth 12 is not biased. Moreover, the standard deviation is practically the

same at all levels of search with only a slight tendency to decrease with increasing search depth. In contrast, standard deviations of the scores are very low at all depths, from which we could assume that $Var_M^A = Var_M^B$ also holds. For better visualization, we only present the mean variance, which as well shows only a slight tendency to decrease with depth. To summarize, taking into account both of these facts, we may conclude that the probability of an error of comparisons performed by CRAFTY at different levels of search is practically the same, and only slightly diminishes with increasing search depth.

3.6 Using Other Programs as Estimators

In Section 3.5, we showed that the probability of an error of comparisons performed by CRAFTY at different levels of search is practically the same, and only slightly diminishes with increasing search depth. The fact that the rankings of the players whose scores are similar to each other fluctuate with depth therefore speaks for the performance of these players, according to the criterion of deviation between the estimator of problem-solving performance and problem solvers, being very similar. Since this is a pioneer work on the subject, no reference exists to support this conclusion.[§] However, this conclusion does not appear surprising bearing in mind that the problem-solvers were all World Chess Champions, so the quality of play in the games at the peak of their careers may indeed not differ significantly when being compared, at least when the majority of the champions is concerned.

Nevertheless, some players whose scores significantly deviate from the others have been established: Capablanca, Euwe, and Steinitz. This can be concluded particularly from the results presented in Table 3.1 and from the fact that Capablanca had the best score in 95% of all the subset-depth combinations in the 100 samples consisted of 500 randomly chosen positions, as mentioned in Subsection 3.3.2. Since the goal of this thesis is *not* to give the final verdict on the question who was the best chess player in history, but merely to determine how can heuristic-search based programs be used as estimators of problem solving performance, we will now focus on these three players and check whether other chess programs rank them in the same order as CRAFTY did.

[§]For an explanation why the widely established ratings do not necessarily reflect the problem-solving performance, refer to Section 2.2.

We will now observe variations with depth of average differences between player's and program's decisions on a large subset of randomly chosen positions from the World Chess Championship matches using three chess programs stronger than CRAFTY. The following programs will be used, namely: RYBKA 2, RYBKA 3, and SHREDDER. The (SSDF) ratings of these programs are given in Table 9.7. Among other observations, the inclusion of the stronger chess programs will help us to deal with another reservation that may be imposed to our methodology of estimating problem-solving performance, namely that the program will give a better ranking to players that have a similar strength to the program itself.

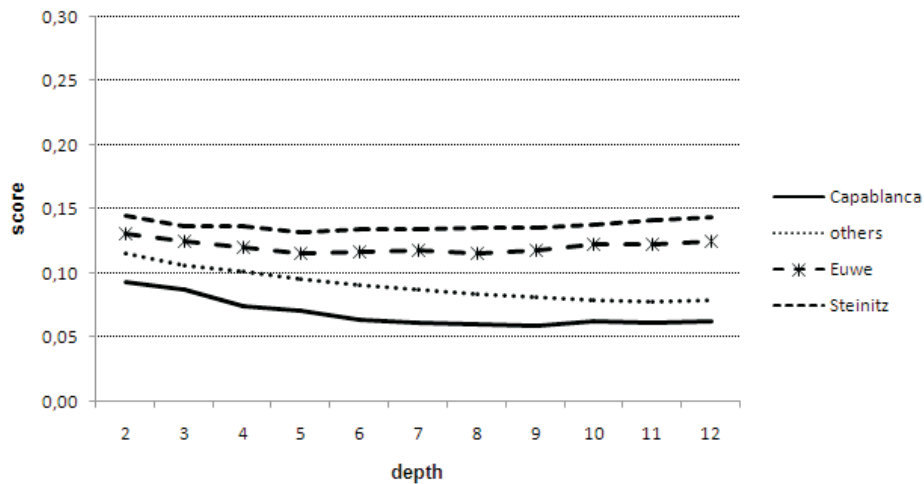


Figure 3.10: The scores of Capablanca, Euwe, Steinitz, and other players, obtained on a large subset of games from World Chess Champion matches, using RYBKA 2.

For each of the three players, we randomly picked at least 1,000 positions for the analysis. As a control group, we randomly picked at least 20,000 positions of the rest of the players. We used the same methodology for determining the rankings according to our basic criterion, that is, the scores at each depth were calculated using Equation 2.4.1.

The results of each of the programs are given in 3.10, 3.11, and 3.12, for RYBKA 2, RYBKA 3, and SHREDDER, respectively. The ranking of Capablanca, Euwe, and Steinitz remained preserved at all depths using any of the programs, and the ranking of the control group is also the same as it was when using CRAFTY. We observed that the three players remained on their positions at any level of search using any of

3. CREDIBILITY OF A HEURISTIC-SEARCH BASED ESTIMATOR

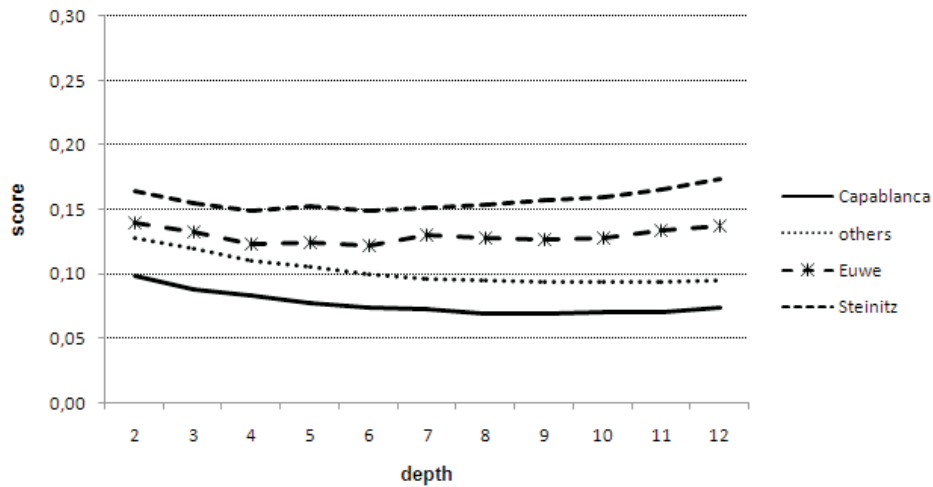


Figure 3.11: The scores of Capablanca, Euwe, Steinitz, and other players, obtained on a large subset of games from World Chess Champion matches, using RYBKA 3.

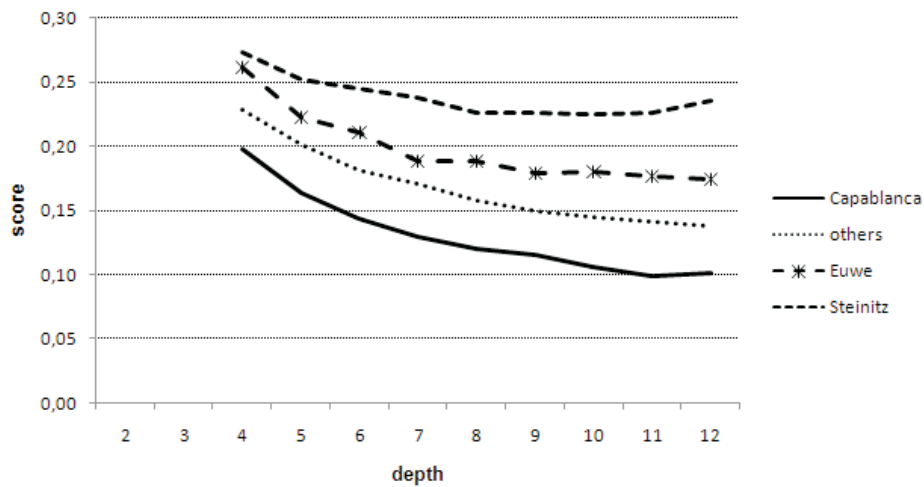


Figure 3.12: The scores of Capablanca, Euwe, Steinitz, and other players, obtained on a large subset of games from World Chess Champion matches, using SHREDDER.

the program, when comparing their scores to the average scores of the players in the control group.

The experimental results presented in 3.10, 3.11, and 3.12 not only confirm that the scores are not invariable for the same program at different depths of search, the scores also differ significantly when using different programs. This is most clearly

seen from Fig. 3.13, where average scores of all the players, obtained on the same large subset of games from World Chess Champion matches, are given for the three programs, and compared to average scores of all the players according to CRAFTY. While the scores of SHREDDER are very similar to the scores of CRAFTY, the scores of the two RYBKAS very much differ.

We would like to emphasize here that the scores obtained by the program only measure the average differences between the players' choices of move and the computer's choice. However, as the analysis shows these scores that are relative to the computer used, have good chances to produce sensible rankings of the players.

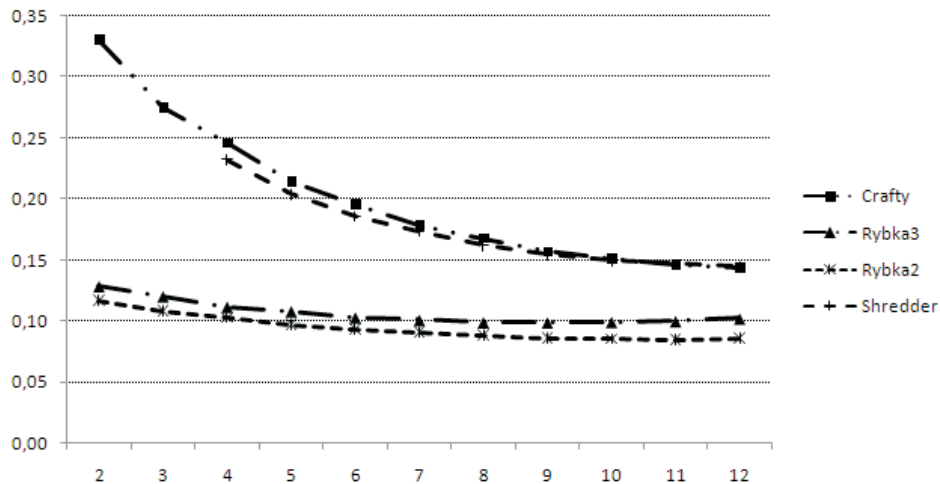


Figure 3.13: Comparison of average scores of all the players, obtained by CRAFTY, SHREDDER, RYBKA 2, and RYBKA 3.

For appropriate interpretation of the obtained scores and rankings of the players, it should be emphasized again that this is only one possible criterion for the comparison of the players among many sensible criteria of very different kinds. In this chapter, we were concerned only with the credibility of the obtained results in the estimates according to this particular criterion. The results show that, at least for the players whose score significantly deviate from the others, the rankings are surprisingly stable over a large interval of search depths, and over a large variation of position sample. Even extremely shallow search of just two or three ply enable reasonable mutual rankings for some pairs of the players.

Last but not least, our experimental findings strongly suggest that in order to

3. CREDIBILITY OF A HEURISTIC-SEARCH BASED ESTIMATOR

obtain a sensible ranking of the players, it is not necessary to use a computer that is stronger than the players themselves.

As we already mentioned, from the chess player's point of view, this score-based criterion is particularly crude in that it does neither take into account the differences in the average difficulty of the positions played by different players, nor does this criterion take into account another important aspect, that is the differences between the playing styles of different players. These topics will be addressed in Chapter 4.

Chapter 4

Assessing Difficulty of Problem Solving Tasks

This chapter is based on the following publication:

1. Guid, M. and Bratko, I. Computer Analysis of World Chess Champions. *ICGA Journal*, Vol. 29, No. 2, pp. 65-73, 2006. [GB06]

The main deficiency of our basic criterion for estimating problem-solving performance, as detailed in the previous two chapters, is in the observation that there are several types of players with specific properties, to whom this criterion does not directly apply. It is reasonable to expect that *positional players* in average commit fewer errors due to the somewhat less complex positions in which they find themselves as a result of their style of play, than *tactical players*. The latter, on average, deal with more complex positions (or more difficult positions in the sense of problem-solving tasks in such positions being harder for humans), but are also better at handling them and use this advantage to achieve excellent results in competition.

We wanted to determine how players would perform when facing equally difficult positions. In other words, we wanted our computer analysis of World Chess Champions to take into account the differences in players' styles to compensate the fact that calm positional players in their typical games have less chance to commit gross tactical errors than aggressive tactical players. Therefore, we designed a method to assess the difficulty of positions.

Although there are enormous differences in the amount of search, nevertheless there are similarities regarding the way chess programs and human players conduct

a search for the best possible move in a given position. They both deal with a giant search tree, with the current position as the root node of the tree, positions that follow with all possible moves as children of the root node, and so on recursively for every node. They both search for the best continuations and doing so, they both try to discard moves that are of no importance for the evaluation of the current position. They only differ in the way they discard them. A computer is running algorithms for efficient subtree pruning whereas a human is depending mainly on his knowledge and experience. Since they are both limited in time, they cannot search to an arbitrary depth, so they eventually have to evaluate a position at one point. They both utilize partial evaluations at given depths of search. While a computer uses evaluations in a numerical form, a human player usually has in mind descriptive evaluations, such as “small advantage”, “decisive advantage”, “unclear position”, etc. Since they may have a great impact on the evaluation, they both check all forced variations (the computer uses *quiescence search* for that purpose) before giving an assessment to the root position. One can therefore draw many parallels between machine and human best-move search procedures, which served as a basis for assessing the complexity of positions.

The basic idea is as follows: a given position is difficult with respect to the task of accurate evaluation and finding the best move, when different “best moves”, which considerably alter the evaluation of the root position, are discovered at different search depths. In such a situation, a player has to analyse more continuations and search to a greater depth from the initial position to find moves that may greatly influence the assessment of the initial position and then eventually choose the best continuation.

As difficulty metric for an individual move, we chose the sum of the absolute differences between the evaluation of the best and the second best move. It is invoked at every time that a change in evaluation occurs when the search depth is increased. A corresponding algorithm for calculating the complexity of a position is given in Algorithm 2. Note however that the aim of this metric is neither to reflect the true cognitive difficulty of a single chess position, nor is it designed for comparing *individual* chess positions in terms of difficulty of their corresponding problem-solving tasks. The aim of this metric is to enable assessing an average difficulty over a sufficiently large set of chess positions, as it was the case in the WCC matches.

The difference between the evaluations of the best and the second-best move represents the significance of change in the best move when the search depth is in-

Algorithm 1 An algorithm for calculating the difficulty of a position.

difficulty := 0;

for depth 2 **to** 12 **do**

if (depth > 2) **and** (previous_best_move **not equals** current_best_move) **then**

 difficulty += |best_move_evaluation - second_best_move_evaluation|

end if

end for

creased. It is reasonable to assume (1) that a position is of higher difficulty, and (2) that it is more difficult to make a decision on a move, when larger changes regarding the best move are detected when increasing search depth. Merely counting the number of changes of the best move at different search depths would give an inadequate metric, because making a good decision should not be difficult in positions where several equally good choices arise.

Graph of errors made by players at different levels of difficulty, shown in Fig. 4.1 clearly indicates the validity of the chosen measure of difficulty of positions; the players made little errors in simple positions, and the error rate increased with increasing difficulty.*

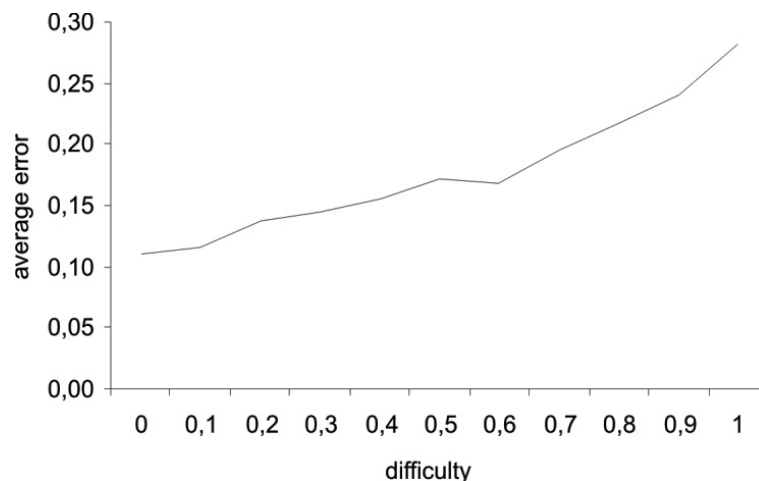


Figure 4.1: Average deviation between evaluations of played moves and best-evaluated moves (errors) made by players at different levels of difficulty.

*Chess program CRAFTY was used in the experiments presented in this chapter.

4.1 Difficulty Measurements in the WCC matches

We applied our difficulty metric to assess an average difficulty of positions of particular players in World Chess Championship matches. The results are shown in Figure 4.2.

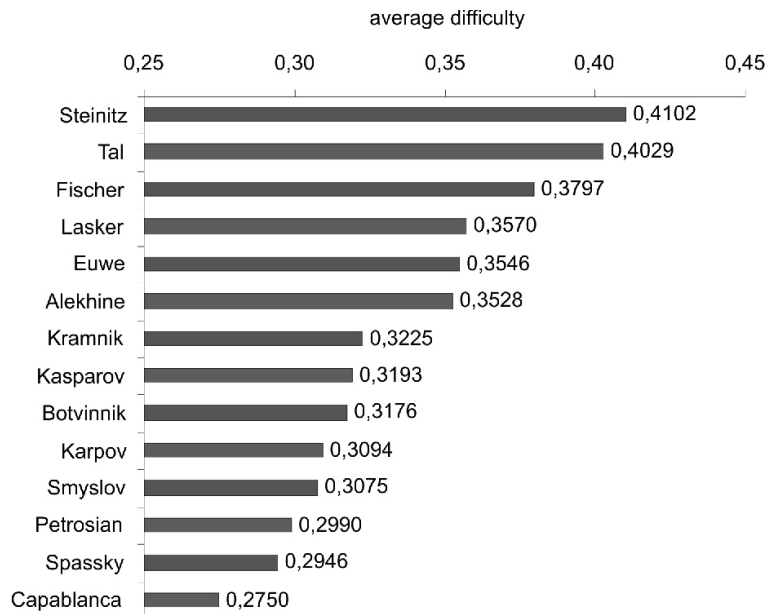


Figure 4.2: Results of average position difficulty measurements for particular World Champions in their World Chess Championship games.

The described metric of position difficulty was also used to determine the distribution of moves played across different intervals of difficulty, based on positions that players had faced themselves. This, in turn, largely defines their *style of play*. In general, Capablanca is renowned for playing a “simple” chess and avoiding complications, while it is common that Steinitz and Tal faced many “wild” positions in their games. The results of the difficulty measurement clearly coincide with this common opinion.

For each player that was taken into consideration, the distribution over difficulty was determined and the average error for each difficulty interval was calculated (numerical scale of difficulty was divided into intervals by steps of 0.1). We also calculated an average distribution of difficulty of moves made for the described intervals for all players combined. Figure 4.3 demonstrates that Capablanca indeed had much less dealings with difficult positions compared to Tal.

The described approach enabled us to calculate an expected average error of

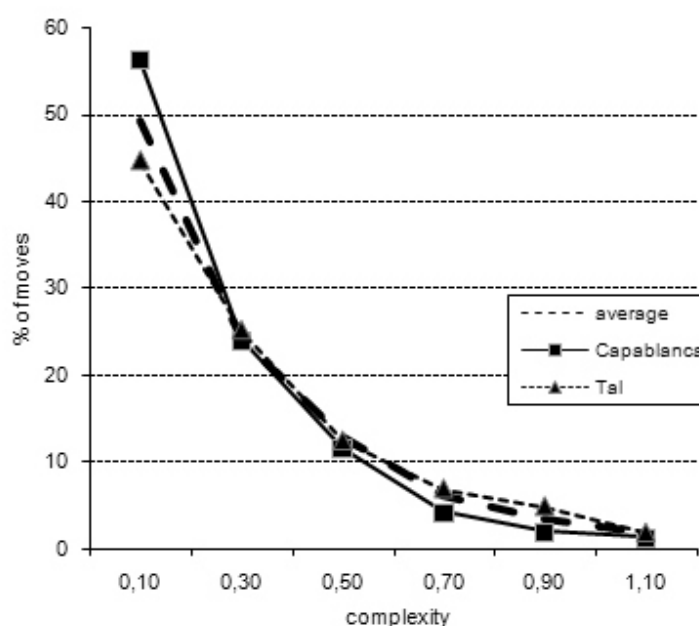


Figure 4.3: Distribution of moves played across different intervals of difficulty.

World Champions in a hypothetical case where they would all play equally difficult positions. We calculated the errors for two cases. Firstly, for a game of average difficulty, averaged among games played by all players. Secondly, for a game of average difficulty, averaged among games played by a single player. The latter represents an attempt to determine how well the players would play, should they all play in the style of Capablanca, Tal, etc. The results of this experiment are given in Figures 4.4 and 4.5.

Distribution of moves in different intervals regarding complexity is closely related with a player's style. Establishing the players' expected errors with a variety of such distributions was another attempt to bring the champions to a "common denominator," by taking into account the differences in their style of play (see Figures 4.4 and 4.5).

Our measure of position difficulty seems to have produced sensible results, which are qualitatively much in line to the observation of how a chess expert would describe the players in this study in terms of their playing style. As a line of future work, it would be interesting to explore by means of a psychological study, how well our difficulty measure reflects the true cognitive difficulty of chess positions.

4. ASSESSING DIFFICULTY OF PROBLEM SOLVING TASKS

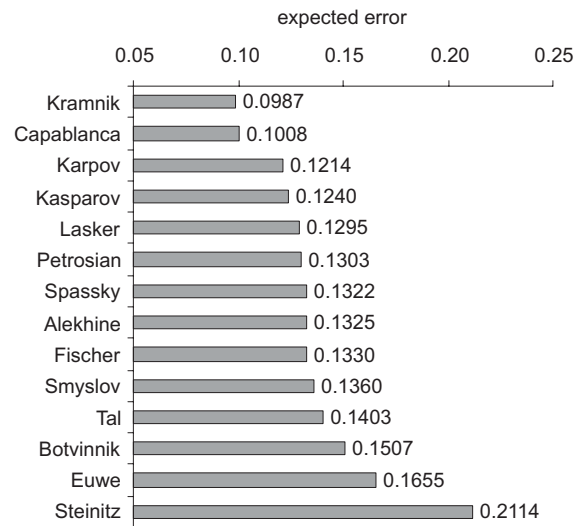


Figure 4.4: How would the players perform if they all had the same distribution of moves played across different intervals of difficulty as Capablanca?

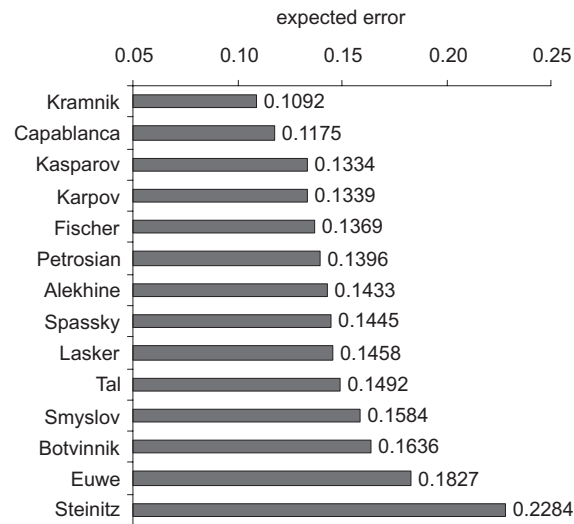


Figure 4.5: How would the players perform if they all had the same distribution of moves played across different intervals of difficulty as Tal?

Part II

Search and Knowledge for Improving Human Problem Solving Performance

Chapter 5

A Heuristic-Search Based Annotator

This chapter is an extensively updated and abridged version of the following publication:

1. Sadikov, A., Možina, M., Guid, M., Krivec, J., and Bratko, I. Automated Chess Tutor. *Computers and Games, Lecture Notes in Computer Science 4630* (eds. Herik, H.J. van den, Ciancarini, P., Donkers, H.H.L.M.(Jeroen)), pp. 13-25. Springer, 2007. [SMG⁺07]

It also includes some materials from the following publication:

1. Guid, M., Možina, M., Krivec, J., Sadikov, A., and Bratko, I. Learning Positional Features for Annotating Chess Games: A Case Study. *Computers and Games, Lecture Notes in Computer Science 5131* (eds. Herik, H.J. van den, Ciancarini, P., Donkers, H.H.L.M.(Jeroen)), pp. 192-204. Springer, 2008. [GMK⁺08]

The second part of the dissertation looks at the topic of Knowledge and Search from a different angle, through the eye of an annotator. The prevailing question is: can a computer act as an expert annotator? In this chapter, we present a roadmap to achieve the goal of building a program for automated commentary of chess games. The current chess software is lacking in this respect. The strong chess programs have long ago achieved the human-grandmaster level of play, yet, they are rather unsuitable for providing instructive commentary. This is logical as they were designed to *play well*, and their “language” is not intended for the annotating purposes. Our aim is to overcome this issue.

5.1 Automatic Annotation of Chess Games

The strong chess programs are though opponents to human grandmasters - already surpassing them in many aspects. In spite of that, their capabilities to *explain* why certain moves are good or bad in a language understandable to humans are quite limited. So far, only some attention was paid to an automatic intelligent annotation of chess games and consequently the progress made in this field is negligible in comparison to the progress in the power of chess engines, which we have witnessed in the last decades. The typical “commentary” in the form of best continuations and their numerical evaluations can hardly be of much help to the chess-player who would like to learn the important concepts that are hidden behind the suggested moves.

In this chapter, we will present a novel approach for automated annotating of chess games. Our approach will be based on a computer program that uses heuristic search. The long-term goal of our research is to develop a computer system that will provide commentary of chess moves and possible continuations in a comprehensible, user-friendly, and instructive way, thus using the power demonstrated by the chess engines for the purposes of annotating. Our approach aims at the following main advantages:

- the ability to annotate chess games during all the phases of the game,
- the automatically generated commentary, aside from the ability to comment on tactical positions, should also express a solid understanding of strategic concepts behind variations that the engines suggest in given positions.

5.1.1 Related Work

The importance of the idea about automatic annotation of chess games was realized long time ago. Around 1980, Donald Michie was probably the first to propose research in this direction. ICGA (then still ICCA) Journal started an annual competition for the best chess annotation program - named after the late Prof. Herschberg. In the mid 1990s, the spirits were high and the progress made even prompted D. Levy and T. Marsland [LM96], members of the competition jury, to write: “If progress in this field continues to be made at the same rate we would expect that, by the end of the century, there will be annotation programs available which can offer a very useful service to chess-players of every strength.” However, none of this have happened

- the 1998 Fritz was the last program to enter the competition, and to this day the commentary of chess programs remains rather spare and is mainly of tactical nature, while the more complex strategic concepts and plans more or less remain unmentioned.

The scientific research in this field was limited only to chess endgames, while the demonstrated concepts all have a common weakness - the inability to extend annotations in practice to the entire game of chess.

Gadwal *et al.* [GGM93] introduced a tutoring system with king, bishop and two pawns on the same file versus lone king endgame. Their approach allows to compile the plans in a given endgame into a strategy graph, which elaborates strategies that students might use as they solve the endgame problem. Later, during tutoring, the strategy graph can be accessed quickly in order to understand a student's moves in terms of his strategies. With such understanding, appropriate knowledge-based commentary is provided to the student. The aim of the proposed approach was not to tutor the whole game of chess, since compiling strategy plans for complex chess middlegames would be unrealistic.

Seidel [Sei94] focused on the lone-king type of elementary endgames such as KRK and KBBK. At the core of the system is a general rule for generating moves for the stronger side that first determines the threats by the weaker side, generates possible actions, and then eliminates those moves that do not parry the threats. Threats and actions represent the chess knowledge of the system and are defined manually. The annotation mechanism follows simply from the move generating scheme by commenting which actions and threats are encountered. While this approach enables specific annotations, the weak point is that a vast amount of manually entered knowledge is required even for simple endgames. Besides, in a commentary on chess games, people are interested in far more than only threats and actions in order to benefit from annotations, in terms of chess knowledge gained.

Herbeck and Barth [HB96] combined alpha-beta search with knowledge in the form of rules. Chess endgames up to four pieces served as their domain. Their algorithm performs a standard search and then follows the principal variation until a position is encountered in which a rule from the knowledge base states the outcome (or a bound on the outcome) of the game. This rule (or a set of them) is then offered to the user as an explanation. Such an approach enabled sensible and instructive commentary, however, the manually constructed rules are hardly appropriate for more complex endgames, let alone for middlegames. As with the other studies mentioned

the main problem with it is the lack of extendability beyond the simple endgames.

5.2 The Annotating System Design

We follow the most natural way of annotating chess games in chess literature. Commentators usually support suggested moves with series of moves (variations), and comment on changes that the given variations would bring and/or describe the envisioned position at the end of the variation. Our main underlying idea is to use heuristic search and chess-program-evaluation-function-like features to describe the changes in the position when a move is made. These elementary features can later be combined to form higher-level concepts understandable to humans. In this manner we bridge the communication barrier between machines and humans. Our annotating system consists of three components.

Search Module

provides principal variations and evaluations that result from heuristic search to various consecutive search depths, for all possible moves in an initial position.

Knowledge Module

provides knowledge in a form of positional-feature values computed for all positions along the obtained principal variations.

Expert Module

uses the principal variations and the positional-feature values to produce instructive and comprehensible annotations, based on a rule-based system.

Search Module guides a computer chess program, that is a chess engine,* to conduct heuristic searches from all possible moves in a given position[†] in order to obtain (a) principal variations as a result of those searches, and (b) corresponding backed-up heuristic evaluations. Preferably the strongest available chess engine and a reasonably high depth of search are used for this purpose, *i.e.*, to assure a high quality of the obtained variations.

*We will from now on often use the term “chess engine” instead of “chess program,” to make the distinction easier between our annotating program and ordinary chess programs (*i.e.*, chess engines).

[†]Even weak moves (according to the engine at the highest search depth) may be interesting for annotating purposes, especially if they seem to be good at lower search depths. In such cases, showing a refutation of a seemingly good move may lead to instructive annotations.

Knowledge Module combines the results of search with potentially useful domain knowledge. The positions along all the principal variations are assigned chess engine's positional feature values.[‡] Additional positional features can also be introduced into the system. We will address obtaining knowledge, in terms of learning positional features useful for annotating chess games, in Chapter 6.

Expert Module uses the results of both other modules to produce comprehensible, user-friendly, and instructive annotations. It is largely based on the rule-based expert system, which will be described in Section 5.4.

5.3 Our Approach to Automatic Annotation

At first glance, moves can be divided into two categories: good moves and bad moves. Yet, when delving into the matter more deeply, one can see that most moves in games, pitting two players with opposing interests against each other, are a sort of tradeoff of positive and negative characteristics of the position. With most moves you gain something *and* you lose something.

These characteristics and their tradeoffs is what our annotating system is calculating and analyzing. For any given move, it calculates what characteristics of the position have changed and on the basis of this and the change in evaluation, as given by the engine, the annotator can elaborate what is the essence of the given move or variation. The general merit of the move, however, is obtained by simply comparing its score with the scores of other possible moves in the given position.

Most chess engines employ an evaluation function in the form of a weighted sum of the position's features. Such features, along with their associated weights, are actually the position's characteristics on which our commenting module is operating. The weights are important too, because they define the relative importance of the features (characteristics).

In the sequel, the mechanisms for calculating positive and negative changes of characteristics are described. Subsection 5.3.1 deals with commenting on why a certain move is good, and Subsection 5.3.2 deals with commenting on why a certain move is bad.

[‡]Actually, positional-feature values used in evaluation function of any chess engine could be used for this purpose. However, it could be beneficial to use positional-feature values of an engine that relies more heavily on the knowledge it contains than on searching deeper. For the method of measuring the quality of knowledge in heuristic evaluation functions, we refer to Section 9.4.

5.3.1 Commenting on Good Characteristics

In general, the annotating system has two options to generate a comment why a certain move is good:

1. the move achieves progress towards some goal, or
2. the move overcomes some weakness or deficiency of the current position.

Let us take a look at both options in more detail.

The basic idea behind the first option is that making headway involves achieving progress towards goals, eventually accomplishing them. The goals in our schema are simply positive changes in the evaluation function's features (or additional positional features, which also describe particular characteristics of the positions). We believe this is a natural way to state simple, comprehensible goals to be achieved. Later we show how the expert system can combine several simple goals into a more structured one thus increasing the expressive power of the annotating system.

Figure 5.1a illustrates the setting for this idea. First, search is employed to obtain a principal variation starting with the move to be commented upon. The final position in the principal variation represents the goal position – this is the position that one can reach from the initial position with the move in question. This position might be viewed as envisioned by the player when he made the move. Second, we calculate which features of the evaluation function have changed and by how much they changed when comparing the envisioned position with the starting position. If one looks at the evaluation function's estimation of a position as a vector of values, this operation is simply a difference between the corresponding position vectors.

The positive characteristics (or rather positively changing characteristics) achieved are those that have positive values in the resulting vector of differences (here we assume that we comment from the white player's perspective; otherwise it is just the opposite). Each such characteristic represents an eventual comment of what the move in question aims to achieve. Basically, at this stage, we obtain a list of positive characteristics which the move (or rather the principal variation starting with the move in question) aims to achieve.

For example, if the following characteristic was singled out as the one that changed positively:

WHITE_KNIGHTS_CENTRALIZATION

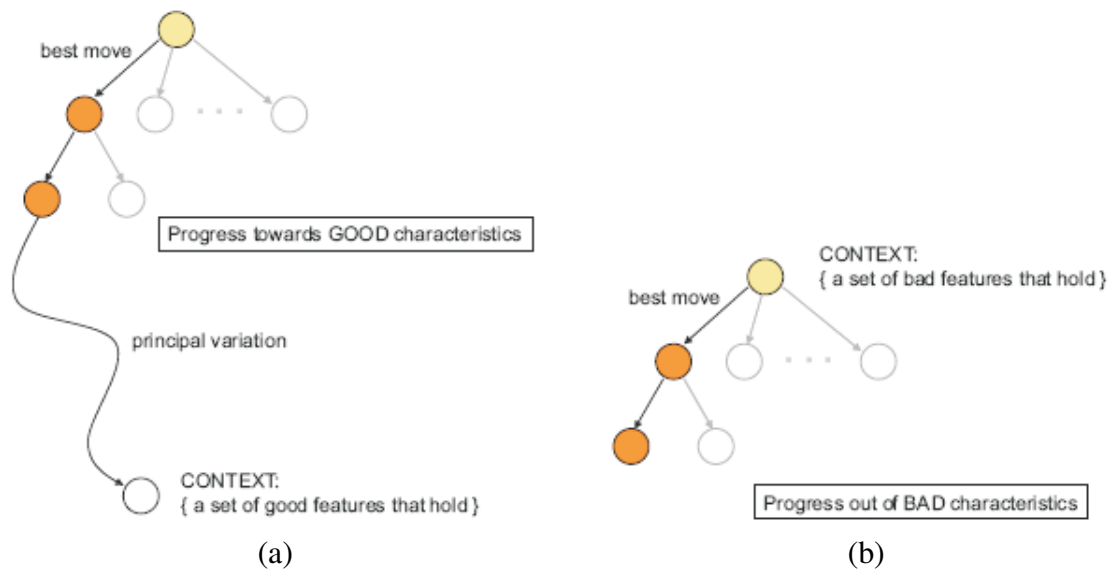


Figure 5.1: Commenting good characteristics

then a possible raw commentary would be “White’s knight is now centralized.” Of course, it is up to the rule-based expert system to decide whether the positive change is big enough actually to produce such comment in a given position, or if it is appropriate to give any annotation in this position at all.

It should be noted that both starting and envisioned position must be quiescent. The starting position should be quiescent, because there is no point in commenting in the middle of a tactical sequence; such a sequence should be viewed as a whole. The envisioned position should be quiescent for obvious reasons – the evaluation based on a non-quiescent position is completely unreliable and thus of little use.

Let us now take a look at the other possibility why a move can be good – namely, because it overcomes some weakness or deficiency of the current position. This situation is illustrated in Figure 5.1b. However, computationally this option is implemented exactly as the first one. Note that good characteristics for the opponent are bad for us, and are represented by negative numbers in the position’s vector. For example, one such negative characteristic (position’s deficiency) could be:

BLACK_BISHOP_PAIR

If the value of this feature changes from some negative value to zero from the start until the end of the principal variation, a possible raw commentary would be “After

[principal variation], Black’s bishop pair is eliminated.” Again, it is up to the rule-based expert system to decide whether to comment or not.

5.3.2 Commenting on Bad Characteristics

The annotating system has three options to generate a comment why a certain move is bad:

1. the move creates a weakness or deficiency in the position,
2. the move spoils some good characteristic of the current position, or
3. the move is compared to a better (the best) possible move.

Options (1) and (2) are quite similar to the two options we encountered earlier when discussing how commenting of good aspects of a move is accomplished. Option(1) mirrors option (1) for generating comments for good moves. The difference is that there we strove to achieve some positive goals, while here the move achieves some negative goal(s). Similarly, option (2) mirrors option (2) for generating comments for good moves. Here, the difference is that instead of removing some weakness or deficiency of the current position, the move removes some positive characteristic of the current position.

From the computational point of view, the only difference is that we are now looking at the evaluation features that are negative in the vector of differences between the starting and envisioned position. The rest is the same. For example, if the following features were flagged as changed for the worse (negatively changing characteristics):

```
BLACK_EVALUATE_PAWNS  
BLACK_ROOK_BEHIND_PASSED_PAWN
```

the possible raw commentaries would be “The move allows the opponent to improve the pawn structure,” and “Black’s rook is now behind a passed pawn.” The rule-based expert system may further refine such comments by stating more concretely how was the pawn structure improved (*e.g.*, the opponent solved his problem of the doubled pawns), and by explaining which rook is behind which passed pawn, if necessary.

However, there is a further difficulty when commenting really bad moves – be it bad in positional or tactical sense. The nature of minimax search is such that it does

not search for a sort of “bad envisioned position”, but rather allows the mistake (as it is forced upon it by the user) and then searches for best play for both sides from that point on. So, in essence, the envisioned position at the end of the principal variation returned by the search may not necessarily reflect the real weakness of the bad move (which is what we would like to comment upon), because this weakness was perhaps traded for some other weakness later in the variation.

Let us illustrate this by an example. Assume that White made a mistake by moving a pawn and thus Black gained a strong outpost for its knight. Later in the principal variation, stemming from this initial pawn move, however, Black exchanged this strong knight for White’s bishop and so eliminated White’s strong bishop pair and doubled White’s pawns. The annotating system, comparing the starting position with the position at the end of principal variation, would comment that “The move allows the opponent to eliminate your bishop pair and to weaken your pawn structure.” While in principle this is true, it may be more in the spirit of annotating to say “The move allows the opponent to gain a strong knight outpost.” The initial comment can prove too abstract to the user. Or, after all, the user can choose not to follow the principal variation at all.

The difficulty we described is actually just a special case of a more general problem – namely, how long should the principal variation be and where in it we should decide to comment. This is a cognitive problem and amongst other things depends on the chess strength of the user (or the audience) for which the system attempts to annotate. In some cases just a single move should be commented, in other cases the whole variation, and in still other cases a part of the variation. The results of searches to lower depths are meant to provide useful information for making annotations of different lengths.

The idea behind option (3) is different from the concepts we discussed so far. Instead of observing what the move achieved, positive or negative, we observe what the move did *not* achieve although it could have. We compare the move played with the best move that was available. In essence, positive and negative changes of characteristics for the move played and for the best move available are calculated for this purpose, and their respective merits compared.

5.4 The Rule-Based Expert System

In principle, the raw comments that could be generated merely by observing changes in particular features along the principal variations can already enable producing relatively intelligent annotations. However, most of the tasks in order to obtain instructive, user-friendly, and comprehensible annotations are left to the expert system. These tasks include solving the following problems:

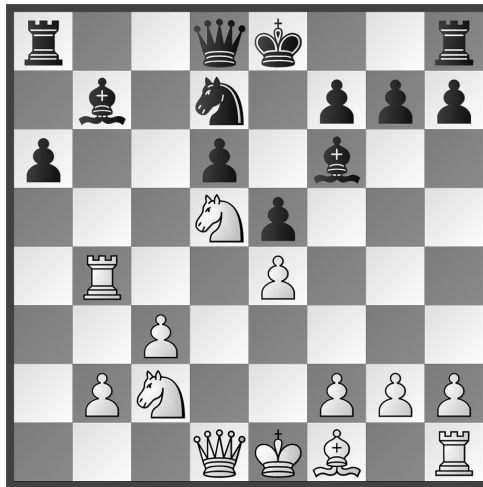
1. deciding when to annotate and when not,
2. choosing which moves (alternatives) to annotate,
3. choosing an appropriate length of displayed variations,
4. deciding what features and feature changes to comment on,
5. deciding to describe feature changes or rather the envisioned position,
6. forming sensible annotations from the calculations of feature changes,
7. refining annotations, making them more instructive and comprehensible.

In Subsection 5.4.1, we describe how our rule-based expert system works. In Subsection 5.4.2 we briefly discuss possibilities for (1) manually changing the obtained rules, and (2) introducing machine learning into the system's development.

5.4.1 Illustrative Example

To illustrate how our expert system works, we present a typical annotation of our annotating program in Figure 5.2. Chess engine CRAFTY was used in the Search Module, and CRAFTY's original positional features (supported with two additional features that measure the activity of the pieces) were used in the Knowledge Module. The commentary on the move 16...Ra7 is shown in the caption. According to the engine, this move is stronger the move played in the game (16...Nc5).

The automated annotator claims that the move 16...Ra7 move leads to (approximately) equal chances for both sides, and supports this assessment with the sequence of moves. The assessment is expressed both qualitatively and numerically (-0.11), the search depth at which it was obtained (13 plies), and the engine used (CRAFTY) are given. The diagram on the left shows the position to be commented upon, while



Kasparov - Shirov, Horgen 1994,
position after 16.Rb4



Position at the end of the envisioned
variation, after 21...e4

Figure 5.2: *Automatic annotation* – “16...Ra7 leads to equality after 17.Bd3 O-O 18.O-O a5 19.Ra4 Nc5 20.Ra3 Bxd5 21.exd5 e4 (-0.11, Crafty:13). Black has improved the pawn structure and the activity of his pieces, White no longer has a strong knight. Black no longer has the advantage of a bishop pair.”

the diagram on the right shows the envisioned position, that is the position at the end of the given variation.

The annotating system decided to produce annotations in this position, since according to the engine a better move (16...Ra7, -0.11) was found then the one that was played in the game (16...Nc5, 0.00). The system therefore commented on the better alternative.

The length of the given variation was shortened by two plies compared to the original principal variation given by CRAFTY (the last two moves, 22.Bc4 a4, were omitted). Note that the length of the variation does not have to be the same as it is in the original principal variation given by the engine at some chosen depth of search. Actually, it is beneficial to shorten the engine’s principal variation, to avoid possible tactical flaws that may occur at the end of it due to a horizon effect (despite having quiescence search enabled). As we mentioned earlier, choosing appropriate length of the variation is a cognitive problem and amongst other things depends on the chess strength of the user (or the audience) the system attempts to annotate for. Of course, the vector of differences between features is always calculated for the variation that the annotating system (not the engine) chooses to display. Choosing automatically the appropriate length of the displayed variation belongs to our future work.

Regarding the depth of search, the depth of 12 or 13 plies usually provides search results of sufficient strength for sensible annotations, when using the chess engine CRAFTY. Of course, some positions (*e.g.*, those of a higher tactical complexity) may demand deeper search. Choosing an appropriate search depth automatically is also a part of our future work.

Tables 5.1 and 5.2 show the calculations of feature-value changes between the diagrammed positions in Figure 5.2. The values are given in “centipawns.” That is, the change of +20, for example, contributes to the positive change of 0.20 in the evaluation by the engine. The flags provide additional useful information to the annotating system. For example, “2 → 1” for the feature `BLACK_BISHOP_PAIR` tells that the number of Black’s bishops reduced from two to one. Additional positional features that were introduced into the annotating system, `WHITE_PIECE_ACTIVITY` and `BLACK_PIECE_ACTIVITY`, are displayed separately from the CRAFTY’s features.

Table 5.1: Vector of negative differences (favorable changes for Black).

Feature	Change	Flag
<code>KING_TROPISM</code>	-52	
<code>WHITE_KNIGHTS_OUTPOSTS</code>	-15	2 → 1
<code>EVALUATE_PAWNS</code>	-15	
<code>BLACK_WEAK_PAWNS</code>	-12	
<code>WHITE_BACK_RANK</code>	-12	
<code>BLACK_PAWN_ADVANCES</code>	-6	
<code>WHITE_KNIGHTS_CENTRALIZATION</code>	-6	2 → 1
<code>BLACK_ROOKS_POSITION</code>	-4	
<code>BLACK_KNIGHTS_CENTRALIZATION</code>	-2	
<code>WHITE_PIECE_ACTIVITY</code>	-10	45 → 35
<code>BLACK_PIECE_ACTIVITY</code>	-4	-36 → -40

The feature `KING_TROPISM`, for example, expresses a difference between a proximity of the pieces of each player to the opponent’s king. Seemingly, this was the most important change to comment on. Obviously, the Black king escaped from the center of the board by castling on the 17th move, and the knight that attacked some squares around the Black’s king was eliminated. Since commentaries like “White’s pieces are now less approximated to the opponent’s king” are really not typical for annotating chess games, giving a comment saying that Black’s king is now safer than

Table 5.2: Vector of positive differences (favorable changes for White).

Feature	Change	Flag
BLACK_BISHOP_PAIR	+20	2 → 1
BLACK_BISHOP_PLUS_PAWNS_ON_COLOR	+16	2 → 1
BLACK_BACK_RANK	+12	
WHITE_BISHOPS_POSITION	+10	
BLACK_BISHOPS_MOBILITY	+9	2 → 1
WHITE_ROOK_HALF_OPEN_FILE	+5	
WHITE_ROOKS_POSITION	+4	
WHITE_PAWN_ADVANCES	+3	
WHITE_BISHOPS_MOBILITY	+3	
BLACK_BISHOPS_POSITION	+2	2 → 1

before may seem to be appropriate. However, was Black’s king really in any danger before? Actually, this attribute alone does not give an answer about the king’s safety. Only combined with another features (such as `EVALUATE_KING_SAFETY`, `BLACK_SAFETY`, or `WHITE_TROPISM`) this feature may be able to help the system to determine whether one side has a pressure against the opponent’s king. In the present case, no rule in the expert system triggered, and no comment was produced based on the changed value of this feature. In general, it is better to not comment at all than giving false annotations.

Let us demonstrate how the annotating system produced the following commentary: “Black has improved the pawn structure.” The values of the following two `CRAFTY`’s features were particularly important in order to induce this comment:

- (a) `EVALUATE_PAWNS`,
- (b) `BLACK_WEAK_PAWNS`.

At the beginning of the variation, the values of these features were 26 (a) and 12 (b), while at the end of the variation they changed to 11 (a) and 0 (b). The positive values here mean that these features favored White. After the relatively big change of the features’ value is detected (compared to other feature-value changes), the annotating system seeks for appropriate rule. In the present case, a rule of the following structure triggered.

```
IF (BLACK_PAWNS < -X) THEN
```

```
IF ( $\Delta$ EVALUATE_PAWNS < -Y) THEN
  IF (( $\Delta$ BLACK_WEAK_PAWNS +  $\Delta$ BLACK_DOUBLED_PAWNS +
    +  $\Delta$ BLACK_PASSED_PAWNS +  $\Delta$ BLACK_ISOLATED_PAWNS +
    +  $\Delta$ BLACK_PAWN_DUO) <= -Z) THEN
    comment('`Black has improved the pawn structure.`')
```

The rule combines several elementary CRAFTY's features to decide whether commenting on improved pawn structure by Black is desirable or not. First, it checks whether there are sufficient Black's pawns on the chessboard, and then how much Black's pawn structure improved over the opponent's pawn structure. Finally, it checks whether the sum of changes of positional feature values that describe Black's pawn structure changed sufficiently in Black's favor. At the relatively simple set rule of rules using this structure for expressing whether one's pawn structure has improved, the values of X , Y , and Z were determined using the expert knowledge only, after the experts became acquainted with the meaning of the feature values that CRAFTY uses. The rule can easily be modified to describe the envisioned position, *e.g.*, by changing the comment to "Black's pawn structure is now improved." It is possible to allow choosing among alternative comments, making the annotations more colorful.

In similar fashion, the other comments were obtained. The comment "White no longer has a strong knight" was made by considering the negatively changed values of `WHITE_KNIGHTS_OUTPOSTS` and `WHITE_KNIGHTS_CENTRALIZATION`, and that the number of White's knights reduced from two to one. For the rest of the feature-value changes, no rule triggered. At last, all the obtained commentary is joint together to form more human-like annotations, as are the ones shown in Figure 5.2.

5.4.2 Possibilities for Manually Changing the Obtained Rules

The elementary positional features that are built in CRAFTY's evaluation function are not always appropriate for successful annotating, while combining them into appropriate rules is often too difficult for domain experts. For example, none of the following attributes that are somewhat associated with the high-level concept of having a pressure against the opponent's king, if used alone in some kind of a rule, would suffice for determining whether the pieces form a threat to the opponent's king: (1) `WHITE_TROPIISM`, (2) `BLACK_SAFETY`, (3) `EVALUATE_KING_SAFETY`, (4) `KING_TROPIISM`. The following comment would sound absurdly: "White im-

proved the safety of his king relatively to the safety of the opponent's king." This is exactly what the feature `EVALUATE_KING_SAFETY` is supposed to express.

Introducing additional elementary positional features into the annotating system usually does not completely solve the problem either, although it may help constructing more powerful rules for successful annotating.

It may seem that relatively complex concepts associated with the game of chess, and numerous exceptions to particular "rules" make this domain inappropriate for applying machine learning techniques, at least for annotating purposes. Indeed it seems rather difficult to express high-level concepts using the limited set of available positional features. However, there is one favorable circumstance: It is not necessary to produce particular annotations, when the annotator is not sure about their correctness. Therefore it is possible to change manually the obtained rules into more demanding ones. For example, by increasing the value of Z in the example rule in Subsection 5.4.1, which would result in demanding greater change in Black's pawn structure improvement in order to comment about it. Moreover, it is not necessary to obtain a "perfect" model for describing some concept – it suffices to obtain at least some rules that can reliably classify new examples, and then use these rules only.

Our approach for formalizing complex positional patterns and thus obtaining rules for describing high-level concepts by using machine learning techniques will be presented in detail in Chapter 6.

5.5 Final Remarks

By providing a roadmap to achieve the goal of building a program for automated commentary of chess games, our goal was to demonstrate that a heuristic-search based program is able to act as an expert annotator of chess games. Our main idea is to use a heuristic-search program to provide results of heuristic search and heuristic-evaluation function's features to describe the changes between the root node and the goal node. The features can then be combined to form higher level concepts understandable to humans. We showed an illustrative example and explained it in detail in order to demonstrate how our annotating program works. However, we view its development as a work in progress – the value of our novel approach still has to be proven. This can be done by submitting our annotating program to the Herschberg Annotation Award Competition, which is one of our future goals.

Chapter 6

Obtaining Knowledge for a Heuristic-Search Based Program

This chapter is an updated and abridged version of the following publication:

1. Guid, M., Možina, M., Krivec, J., Sadikov, A., and Bratko, I. Learning Positional Features for Annotating Chess Games: A Case Study. *Computers and Games, Lecture Notes in Computer Science 5131* (eds. Herik, H.J. van den, Ciancarini, P., Donkers, H.H.L.M.(Jeroen)), pp. 192-204. Springer, 2008. [GMK⁺08]

It also includes some materials from the following publication:

1. Možina, M., Guid, M., Krivec, J., Sadikov, A., and Bratko, I. Fighting Knowledge Acquisition Bottleneck with Argument Based Machine Learning. *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pp. 234-238. IOS Press, 2008. [MGK⁺08]

In this chapter, we investigate what type of knowledge is necessary for the task of an expert annotator. We point out certain differences between the computer programs which are specialized for playing chess and a program which is aimed at providing quality commentary. A link with machine learning techniques is proposed for the elicitation of knowledge from human experts. After examining several opportunities for introducing machine learning into development of a chess annotating system as it was introduced in Chapter 5, we embark upon argument-based machine learning

as a method of choice for obtaining useful knowledge for a heuristic-search based annotating program.

6.1 Knowledge Acquisition for Chess Annotation

In this chapter, we investigate a particular aspect in the development of a chess annotating software - the ability of making intelligent comments on the positional aspects of a chess game. This task is made more difficult by the fact that the strength of the chess playing programs mainly comes from search and not from subtle positional knowledge which is necessary for generating positional comments. Therefore, components of a chess program's evaluation function are not sufficient for making in-depth positional comments. Defining deep positional patterns requires powerful knowledge-elicitation methods.

Our approach to the generation of positional comments makes use of elements of a chess evaluation function. However, more sophisticated positional patterns have to be introduced in addition to the features contained in an evaluation function. Defining such sophisticated positional patterns is often a difficult knowledge-elicitation task.

In the presented case study, we consider the elicitation of the well-known chess concept of the *bad bishop*. There is a general agreement in the chess literature and among chess players about the intuition behind this concept. However, formalizing it in a way that would enable an annotating system to decide reliably whether a bishop in a given position is bad turned out to be quite difficult even for chess experts.

To alleviate the knowledge-elicitation problem, we applied the recently developed approach of Argument Based Machine Learning (ABML) [MvB07]. The efficacy of ABML comes from its unique ability to make use of expert's arguments (or justifications) for selected example cases. This approach is natural and effective for the expert because he can concentrate on explaining concrete cases, and does not have to construct general rules, which is much more difficult. The method also leads the expert to think about new relevant descriptive features, thereby improving the description language that the learning program uses.

Through a case study, we present an application of argument-based machine learning to the construction of more complex positional features, in order to provide our annotating system (see Chapter 5) with an ability to comment on various positional intricacies of positions in the game of chess.

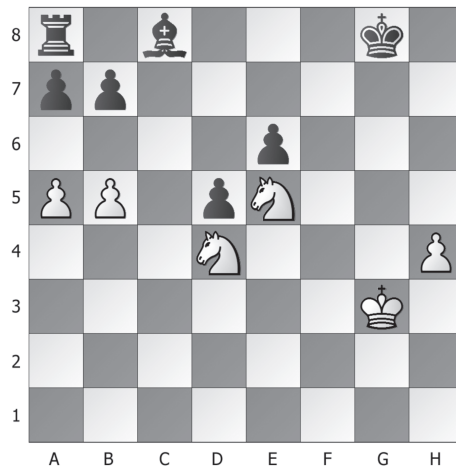


Figure 6.1: Classical example of a bad bishop.

6.2 Positional Features for Annotating Chess Games

The programs that are aimed to *play* chess successfully do not need to be aware of all the concepts that would otherwise be useful for giving instructive annotations. There is always a dilemma how much knowledge to implement into evaluation functions of the programs in order to achieve best tournament performances. The more knowledge means less time for efficient search and vice versa. It is commonly known that some programs have more knowledgeable evaluation functions, while others rely more on efficient search algorithms that allow them to reach higher search depths.

To illustrate our points, we will introduce a concept of the *bad bishop*. Watson [Wat99] gives the following definition as traditional: a bishop that is on the same color of squares as its own pawns is bad, since its mobility is restricted by its own pawns and it does not defend the squares in front of these pawns. Moreover, he puts forward that centralization of these pawns is the main factor in deciding whether the bishop is bad or not. In the middlegame, he continues, the most important in this aspect are d and e pawns, followed by c and f pawns, while the rest of the pawns on the same color of a square as the bishop, are irrelevant (up to the endgame, where they might again become an important factor for determining the goodness of the bishop).

The example in Figure 6.1 is taken from the classic book by Aaron Nimzovich, *The Blockade* [Nim06]. The black bishop on c8 is bad, since its activity is significantly hindered by its own pawns. Furthermore, these pawns are blockaded by the pieces of his opponent, which makes it even harder for black to activate the bishop.

6. OBTAINING KNOWLEDGE FOR A HEURISTIC-SEARCH BASED PROGRAM

Chess program CRAFTY, for example, has several positional features that are associated with the goodness of the bishops, but they are insufficient fully to describe this concept. They apply to both bishops for one side at the same time, *i.e.*, the values for both bishops are represented by one feature only. Even if we arrange to obtain the feature values for each bishop separately, these positional features are still not appropriate to describe the goodness of a bishop, with the aim to annotate chess games in an instructive way.

Table 6.1 shows CRAFTY's most relevant positional features for describing a bad bishop. As it becomes clear from the descriptions of these features and their deficiencies from the purpose of describing bad bishops, CRAFTY clearly could not be aware of such a concept. For example, if we move pawns e6 and d5 to g6 and h7 (preserving the value of BLACK_BISHOP_PLUS_PAWNS_ON_COLOR - since pawns on the same color of the square as the bishop carry the same penalty, regardless of their position) and the rook from a8 to d7 (the value of BLACK_BISHOPS_MOBILITY even decreases, as the bishop is attacking one square less), the bishop clearly would not be bad, but in CRAFTY's view it would be even worse than in the given position.

Table 6.1: Some CRAFTY's positional features that have a potential for describing bad bishops and their deficiencies for doing so successfully.

Feature	Description	Deficiency for annotating
BLACK_BISHOP_PLUS_PAWNS_ON_COLOR	a number of own pawns that are on the same color of the square as the bishop	all such pawns count the same, regardless of their position and how badly they restrict the bishop
BLACK_BISHOPS_POSITION	an evaluation of the bishop's position based on predefined values for particular squares	such predefined value is not the actual value of the bishop's placement in a particular position
BLACK_BISHOPS_MOBILITY	the number of squares that the bishop attacks	the number of attacked squares and the actual bishop's mobility are not necessarily the same thing

In order to introduce a new positional feature (say BAD_BISHOP) that would allow commenting on such complex concepts, as is the concept of the bad bishop, it is therefore essential first to obtain additional (simple) positional features, and then combining them into some kind of rules that would allow to obtain the value of the

new (more complex) positional feature `BAD_BISHOP`.

As we mentioned in Subsection 5.4.2, it is not necessary to comment on particular feature each time it occurs. When the annotator is not sure about some feature in the position, it is better to say nothing at all than giving false comments.

6.2.1 The Static Nature of Positional Features

Positional features are static in their nature - they describe the state of their purposed issue for the current position only. It is heuristic search that enables them to fulfil their purpose - contributing to the program finding the best moves. For example, in position in Figure 1, if we moved the knight from e5 to h1, and decided that black is to move, black would easily solve all his problems by playing e6-e5, chasing the other white knight away and freeing both of his pieces. The positional features from Table 6.1, among others, would contribute to deciding for this freeing move, since the values of all three attributes become more desirable soon along the principal variation (*e.g.*, after e6-e5 and Bc8-f5, there are less pawns on bishop's square color, and the bishop itself is placed on a square with a higher predefined value and also attacks more squares). Although the mentioned positional features are not suitable for commenting on the bad bishop, they nevertheless help it to become a good one.

It is also desirable for positional features for annotating chess games to be of static nature. For example, it is up to the chess engine to determine whether the freeing move e6-e5 is possible or not (*e.g.*, in case of white king on f4 and the e5 knight still on h1 it would drop at least a pawn).

6.3 Application of Machine Learning Techniques

In the sequel of this chapter, we demonstrate the construction of a static positional feature, `BAD_BISHOP` (with possible values *yes* or *no*), which was designed for commenting on bad bishops (possibly combined with some heuristic search).

In our domain, it turns out to be extremely difficult for a chess expert to define appropriate rules, using typical positional features, for the program to be able to recognize complex concepts, such as *the bad bishop*. Our domain experts* defined the rules, using CRAFTY's positional features only, which in their opinion described bad

*WGM Jana Krivec and FM Matej Guid.

bishops in the best possible way (considering the constraint of having only CRAFTY's features at disposal). The rules were of the following type:

```
IF ( |BLACK_BISHOP_PLUS_PAWNS_ON_COLOR| > X)
    AND ( |BLACK_BISHOPS_MOBILITY| < Y)
THEN BAD_BISHOP = yes
```

Three such rules were given, depending on the number of black pawns in the position. The positional features and the values for X and Y were determined, after the experts had become acquainted with the exact meaning of CRAFTY's positional features and observed their values in several positions of various types. However, after examining the outcome of these rules on various chess positions, it turned out that the rules performed rather poorly, which was the key motivation for introducing machine learning into the system's development.

In Subsection 6.3.1, we describe how the learning data set was obtained. In Subsection 6.3.2, we present the results of using typical machine learning methods. Finally, in Subsection 6.3.3, we introduce Argument Based Machine Learning as our method of choice for the acquisition of knowledge.

6.3.1 The Learning Data Set

The learning data set consisted of middlegame positions[†] from real chess games, where the black player has only one bishop. Based on the aforementioned expert-crafted rules, positions were obtained automatically from a large database of chess games. In all positions, the quiescence criterion was satisfied. The bishops were a subject of evaluation by the experts.

When chess experts comment on concepts such as the bad bishop, they also have dynamic aspects of a position in mind. Therefore, assessing bishops "statically" is slightly counter-intuitive from the chess-player's point of view. After a careful deliberation, the following rules were chosen for determining a bad bishop from the static point of view.

The bishop is *bad* from the static point of view in some position, if:

1. its improvement or exchange would notably change the evaluation of the position in favor of the player possessing it,

[†]While the concept of the bad bishop hardly applies to the early opening phase, different rules for determining bad bishops apply in the endgames (see Watson's definition in Section 6.2).

2. the pawn structure, especially the one of the player with this bishop, notably limits its chances for taking an active part in the game,
3. its mobility in this position is limited or not important for the evaluation.

These rules seem to be in line with the traditional definition of the bad bishop, and in the experts' opinion lead to sensible classification - in positions where assessment from the static point of view differs from the one obtained from the usual (dynamic) point of view, it seems very likely that a possible implementation of heuristic search, using the newly obtained positional feature `BAD_BISHOP` (such search could enable the program to realize whether the bishop is more than just temporarily bad and thus worth commenting on), would lead to sensible judgement on whether to comment on the bad bishop or not.

The learning data set consisted of 200 positions[‡]. We deliberately included notably more bishops labeled as “bad” by the initial rules given by the experts, due to our expectations (based on the unsuitability of CRAFTY's positional features) that many of the bishops labeled as “bad” will not be assessed so after the experts' examination of the positions. After examination by the experts, 80 examples in the data set obtained the class value *yes* (“bad”) and 120 examples obtained the class value *no* (“not bad”). It turned out that only 59% of the examples were correctly classified by the expert-crafted rules.

6.3.2 Using Ordinary Machine Learning Methods

As the expert-crafted rules scored only 59% classification accuracy on our data set, which is clearly insufficient for annotating purposes, there is a clear motivation for the use of machine learning. However, as classification accuracy equally penalizes false positives (“not bad” classified as “bad”) and false negatives, we should also use precision, which measures the percentage of true “bad” bishops among ones that were classified as “bad”. Remember, falsely commenting is worse than not commenting at all.

From the many available machine learning methods we decided to take only those that produce understandable models, as it will be useful later to be able to give an explanation why a bishop is bad and not only labeling it as such. We chose standard

[‡]This number was chosen as the most feasible one, considering limited available time of the experts. The quality of the final model implies that the number of selected positions in the learning data set was high enough.

machine learning methods given in Table 6.2. We also give accuracy and precision results of these methods on our learning set.

Table 6.2: The machine learning methods' performance with CRAFTY's features.

Method	Classification accuracy	Precision
Decision trees (C4.5)	71%	64%
Logistic regression	80%	76%
Rule learning (CN2)	73%	75%

All the accuracy and precision results were obtained through 10-fold cross validation. All the methods achieve better accuracies than expert given rules, but they are still too inaccurate for commenting purposes.

6.3.3 Argument Based Machine Learning

Argument Based Machine Learning (ABML, [MvB07]) is machine learning extended with some concepts from argumentation. Argumentation is a branch of artificial intelligence that analyses reasoning where arguments for and against a certain claim are produced and evaluated [PV02].

Arguments are used in ABML to enhance learning examples. Each argument is attached to a single learning example only, while one example can have several arguments. There are two types of arguments; positive arguments are used to explain (or argue) why a certain learning example is in the class as given, and negative arguments are used to explain why it should not be in the class as given. We used only positive arguments in this work, as negatives were not required. Examples with attached arguments are called *argumented examples*.

Arguments are usually provided by domain experts who find it natural to articulate their knowledge in this manner. While it is generally accepted that giving domain knowledge usually poses a problem, in ABML they need to focus on one specific case only at a time and provide knowledge that seems relevant for this case and does not have to be valid for the whole domain. The idea can be easily illustrated with the task of commenting on chess games. It would be hard to talk about chess moves in general to decide precisely when they are good or bad. However, if an expert is asked to comment on a particular move in a given position, he will be able to offer an ex-

planation and provide relevant elements of this position. Naturally, in a new position the same argument could be incorrect.

An ABML method is required to induce a theory that uses given arguments to explain the examples. Thus arguments constrain the combinatorial search among possible hypotheses, and also direct the search towards hypotheses that are more comprehensible in the light of expert's background knowledge. If an ABML method is used on normal examples only (without arguments), then it should act the same as a normal machine learning method. We used the method ABCN2 [MvB07], an argument-based extension of the well known method CN2 [CB91], that learns a set of unordered probabilistic rules from argumented examples. In ABCN2, the theory (a set of rules) is said to explain the examples using given arguments, when there exists at least one rule for each argumented example that contains at least one positive argument in the condition part.

In addition to rules we need an inference mechanism to enable reasoning about new cases. Given the nature of the domain, we decided to learn only rules for "bad" bishop and classify a new example as "bad" whenever at least one of the learned rules triggered.

Asking experts to give arguments to the whole learning set is not likely to be feasible, since it requires too much time and effort. The following loop describes an iterative process for acquiring arguments and new attributes from experts.

Step 1 Learn a set of rules.

Step 2 Search for problematic cases in the data set; these are the examples that are misclassified by the induced rules.

Step 3 If no problematic examples are found, stop the process.

Step 4 Select a problematic example and present it to experts. If the case is a position with a "bad" bishop, then experts are asked to explain why this bishop is "bad". If it is a "not bad" bishop position, then we search for the culpable rule predicting "bad" and ask experts to explain an example with the class value *yes* ("bad") from the set of examples covered only by this rule. In the latter case experts need to be careful to provide reasons that are not true in the problematic position. Problematic positions with a "not bad" bishop are called *counter-examples*.

Step 5 Experts have three possibilities of responding to the presented case.

1. They can give reasons why bishop is “bad”. Reasons are added to the example in the data set.
2. If they cannot explain it with given attributes, they can introduce a new attribute (or improve an existing one), which is then added to the domain.
3. Experts can decide that this bishop is actually “good” and thus the class of the example needs to be changed.

If experts are unable to explain the example, we select another one.

Step 6 Return to step 1.

In the sequel, we present in detail the interactive procedure between the domain expert(s) and ABML during knowledge acquisition.

6.4 Knowledge Elicitation Process

Table 6.3 shows the three rules induced in the first iteration of the aforementioned process, where only CRAFTY’s positional features were used and no arguments have been given yet. Condition part of a rule is the conjunction of the features indicated in the corresponding column. In the cases with no threshold specified, the feature is not part of the corresponding rule. For example, the rule #1[§] is:

```
IF BLACK_BISHOP_MOBILITY > -12 THEN BAD_BISHOP = yes
```

The distribution of positive and negative examples covered by each of the rules speaks about the relatively poor quality of these rules - especially in the last rule.

Figure 6.2 (left) shows the first problematic example selected by our algorithm. The experts were asked to describe why the black bishop is bad. Based on their answer, another attribute, BAD_PAWNS, was added into the domain. The experts designed a look-up table (see Figure 6.2 (right)) with predefined values for the pawns that are on the color of the square of the bishop in order to assign weights to such

[§]The negative values of BLACK_BISHOP_MOBILITY are the consequence of CRAFTY using negative values for describing features that are good for Black. The more squares this bishop is attacking, the more negative this value. For each attacked square, the feature value is decreased by -4. For example, the value of -12 means that the bishop is attacking three squares.

Table 6.3: The rules for `BAD_BISHOP = yes`, after the first iteration of the process.

Positional feature	#1	#2	#3
<code>BLACK_BISHOPS_MOBILITY</code>	> -12	> -18	> -18
<code>BISHOP_PLUS_PAWN_ON_COLOR</code>			> 12
<code>BLACK_BISHOP_POSITION</code>		> 4	
positive examples (“bad”)	40	44	67
negative examples (“not bad”)	4	7	25

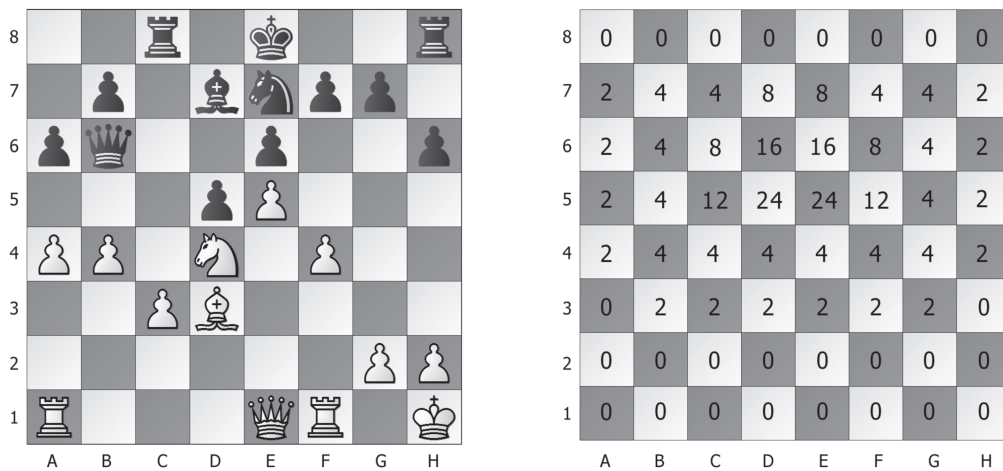


Figure 6.2: The experts were asked the question: “*Why is the black bishop bad?*” They used their domain knowledge to provide the following answer: “*The black bishop is bad, since a lot of black pawns are on the same color as the bishop. Especially the central pawns notably limit its chances for taking an active part in the game.*” The need for the attribute `BAD_PAWNS` was identified. The experts designed a look-up table with predefined values for the pawns that are on the color of the square of the bishop in order to assign weights to such pawns.

pawns. According to Watson’s definition, centralization of the pawns has been taken into account. Several other attributes that were added at later stages (see Table 6.4) used this look-up table to assess heuristically an influence of such bad pawns on the evaluation of the bishop.

Table 6.4 presents the list of attributes that were added to the domain during the process. To give an example of how the values of these new attributes are obtained, we will calculate the value of the attribute `BAD_PAWNS_AHEAD` for the position in Figure 2. This attribute provides an assessment of pawns on the same color of square as the bishop that are in front of it. There are three such pawns in that position: e6,

6. OBTAINING KNOWLEDGE FOR A HEURISTIC-SEARCH BASED PROGRAM

Table 6.4: The new attributes, and iterations when they were added to the domain.

Attribute	Description	It.
BAD_PAWNS	pawns on the color of the square of the bishop - weighted according to their squares (<i>bad pawns</i>)	2
BAD_PAWNS_AHEAD	bad pawns ahead of the bishop	3
BLOCKED_DIAGONAL	bad pawns that block the bishop's (front) diagonals	4
BLOCKED_BAD_PAWNS	bad pawns, blocked by opponent's pawns or pieces	5
IMPROVED_BISHOP_MOBILITY	number of squares accessible to the bishop, taking into account only pawns of both opponents	6
BLOCKED_PAWNS_BLOCK_DIAGONAL	bad pawns, blocked by opponent's pawns or pieces, that block the bishop's (front) diagonals	12

d5, and a6. For each of these pawns their corresponding values are obtained from the look-up table, that is, 16, 24, and 2, respectively. The sum of these values ($16 + 24 + 2 = 42$) represents the value of the attribute `BAD_PAWNS_AHEAD` in that position.

Figure 6.3 shows an example how the argument given to some particular position could be improved by the expert, using some help by the machine learning method, which automatically suggests him appropriate counter-example. The counter-examples are another effective feature for overcoming the knowledge acquisition bottleneck.

The final rules at the end of the process are presented in Table 6.5 (for the interpretation of this presentation, see the description before Table 6.3). The obtained rules for the new positional feature `BAD_BISHOP` only cover positive examples, have a pure distribution (no misclassified examples), and also appear sensible to the experts.

Particularly valuable is that the rules enable not only commenting on whether a bishop is bad, but also *why* it is bad. Formulation of the explanations is provided in the expert module of our annotating system. For example, if the rule #2 from Table 6.5 triggers in a particular position, the following comment could be given: “*Black bishop is bad, since black pawns on the same color of squares ahead of it, and pawns of both opponents restrict its mobility.*”

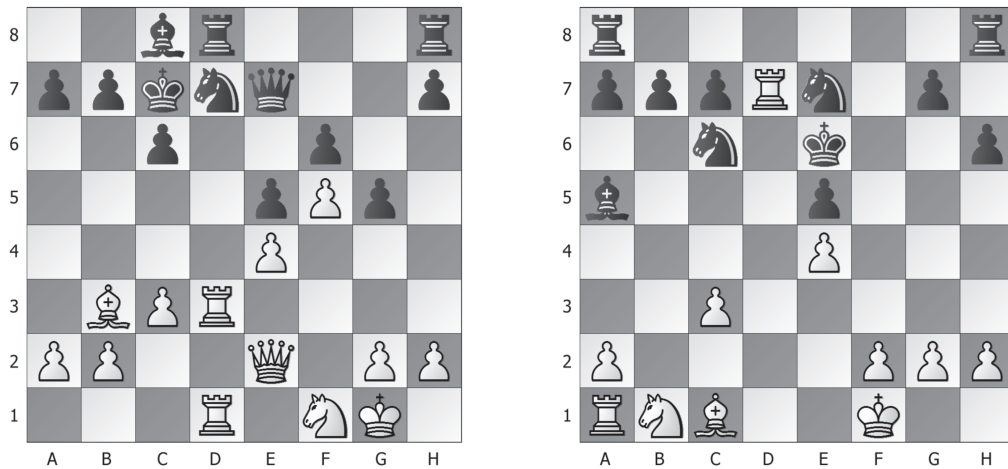


Figure 6.3: After iteration 6, the expert gave the following description why the bishop is bad in position on the left: “*The bishop is bad, because, taking the pawn structure into account, only one square is accessible to it.*” The argument “IMPROVED_BISHOP_MOBILITY=low” was added to this position, based on this description. However, in the next iteration, the machine learning method selected the position on the right, where the bishop is classified as “not bad”, as the counter-example. After the expert’s examination, the following significant difference between the two positions was determined: in the position on the right, there are no bad pawns ahead of the bishop. Based on that, the argument to the position on the left was improved to “IMPROVED_BISHOP_MOBILITY=low AND BAD_PAWNS_AHEAD=high”.

6.5 Assessment and Discussion

The machine learning methods that were used on CRAFTY’s original positional feature values were tested on the same data, supplemented with the newly obtained attribute values. All the accuracy and precision results were again obtained through 10-fold cross validation.

The results are presented in Table 6.6. They suggest that the performance of other algorithms could also be improved by adding appropriate additional attributes (compare to Table 6.2). However, using arguments (as with the method ABCN2), besides stimulating the expert to identify the need for useful additional attributes, also guides the method towards appropriate combinations of attributes, which is likely to lead to even more accurate models.

The main advantage of ABML over classical machine learning is the ability to take advantage of an expert’s prior knowledge in the induction procedure. This leads

6. OBTAINING KNOWLEDGE FOR A HEURISTIC-SEARCH BASED PROGRAM

Table 6.5: The rules for `BAD_BISHOP = yes`, obtained after the 14th (final) iteration.

Positional feature	#1	#2	#3	#4	#5	#6	#7
BAD_PAWNS					> 14		> 32
BAD_PAWNS_AHEAD	> 20	> 18		> 26	> 28	> 12	
BLOCKED_DIAGONAL	> 4		> 16				> 16
BLOCKED_BAD_PAWNS				> 0			
IMPROVED_BISHOP_MOBILITY		< 3	< 4	< 4		< 2	< 5
BLOCKED_PAWNS_BLOCK_DIAGONAL					> 0		
BLACK_BISHOPS_MOBILITY	< -15						
positive examples	46	46	42	38	38	36	31
negative examples	0	0	0	0	0	0	0

Table 6.6: The machine learning methods' performance with the data, supplemented with the newly obtained attribute values.

Method	Classification accuracy	Precision
Decision trees (C4.5)	85%	85%
Logistic regression	89%	91%
Rule learning (CN2)	91%	94%
Rule learning with arguments (ABCN2)	94%	96%

to hypotheses comprehensible to experts, as it explains learning examples using the same arguments as the expert did. In our case study this was confirmed by chess experts. According to them, the final set of rules are more alike to their understanding of the bad bishop concept than the initial rules were. Furthermore, the final rules were also recognized to be in accordance with the traditional definition of a bad bishop.

Our domain experts clearly preferred the ABML approach to manual knowledge acquisition. The formalization of the concept of bad bishop turned out to be beyond the practical ability of our chess experts (a master and a woman grandmaster). They described the process as time consuming and hard, mainly because it is difficult to consider all relevant elements. ABML facilitates knowledge acquisition by fighting these problems directly. Experts do not need to consider all possibly relevant elements, but only elements relevant for a specific case, which is much easier.

Moreover, by selecting only critical examples, the time of experts involvement is decreased, making the whole process much less time consuming.

Given our experimental findings in the domain of chess, we believe that our approach to knowledge elicitation based on the ABML type of machine learning will be most helpful for obtaining positional features useful for heuristic-search based programs. Complex features such as `BAD_BISHOP` that we formalized in our case study may not always be suitable for evaluation functions of competitive programs, where time spent on heuristic search is quite important (besides, appropriate weights of these features should be determined). Nevertheless, they could serve well for annotation purposes. For example, when used in a software such as the chess annotating system presented in Chapter 5.

Chapter 7

Deriving Concepts and Strategies from Chess Tablebases

This chapter is an updated and abridged version of the following publication:

1. Guid, M., Možina, Sadikov, A., and Bratko, I. Deriving Concepts and Strategies from Chess Tablebases. *ACG 2009, Lecture Notes in Computer Science 6048* (eds. Herik, H.J. van den, and Spronck, Peter), pp. 195-207. Springer, 2010. [GMSB10]

In this chapter, we demonstrate a semi-automatic procedure for deriving concepts and strategies usable for intelligent tutoring purposes from chess tablebases, *i.e.*, databases that contain perfect information. We focus on constructing human-friendly textbook instructions for teaching a difficult-to-master KBNK (king, bishop and knight versus a lone king) chess endgame. The instructions were obtained from a hierarchical model of semi-automatically generated goal-based rules, and represent the knowledge of a search-based program that is capable of giving instructive annotations.

The chapter is organized as follows. Section 7.1 contains a short overview of related work on extracting knowledge from chess tablebases. In Section 7.2, we first present the obtained textbook instructions. The hierarchical model of ordered set of rules is given at the end of the section, together with an example game containing automatically generated instructions based on these rules. In Section 7.3, we introduce the basics of our approach to goal-based rule learning, explain the guidelines for interaction between the machine and the expert in order to obtain a human-

understandable rule-based model for playing a chess endgame, and describe how the instructions for KBNK were derived semi-automatically from our hierarchical rule-based model. In Section 7.4, we present the evaluation of the instructions by three renown chess teachers, and an evaluation of human-like style of play generated by our method by four international grandmasters. We conclude the chapter by some final remarks and intentions for further work.

7.1 Learning from Perfect Information

Chess tablebases [Tho86] have enabled people a glimpse of how a perfect play looks like. It seems, however, that people are ill adapted to understanding this perfection. While tablebases are of enormous help to computers, people are for the most part puzzled by the style of play generated by tablebases. Yet, people would very much like to learn as much as possible, and there is no doubt that tablebases contain an enormous amount of potential knowledge – but in a form not easily accessible to a human mind.

There have been many attempts to extract knowledge from tablebases. Perhaps two best documented examples are a research project carried out by a chess study specialist John Roycroft [Roy88] and the work of grandmaster John Nunn resulting in two books on pawnless endings [Nun95; Nun02]. All the attempts, however, had at most limited success.

The goal of Roycroft's study was that he would learn himself reliably to play the KBBKN endgame (king and two bishops vs. king and knight). This endgame was for a long time considered generally to be drawn, until the KBBKN tablebase was computed. The tablebase showed that the side with two bishops can usually force a win, but the winning play is extremely difficult and takes a long sequence of moves under optimal play by both sides. Many moves in the optimal play for the stronger side are completely obscure to a human. Roycroft tried to extract a human-executable winning strategy by the help of this tablebase, trying manually to discover important concepts in this endgame which would enable a human to win reliably. After a one year's effort, the project ended with rather limited success when Roycroft's accumulated skill for this endgame was still not quite sufficient to win actually against the tablebase in many of the won KBBKN positions.

The task of learning is not any easier for the computer. In an overview of the learning methods in games, Fürnkranz indicated that machine learning of understand-

able and usable concepts over the years did not yield much success [FÖ1]. There have been various attempts to bridge the gap between perfect information stored in tablebases and human-usable strategies. While some of these approaches succeeded for relatively small domains (such as KRK endgame in chess), the resulting models are hardly intelligible to human experts [vdHUvR02], not to mention beginners and novices. All related work did not result in breakthroughs in more complex domains. Moreover, the research questions how to learn human-understandable models and use them to generate instructions suitable for teaching humans remained open.

In a way learning from tablebases resembles closely the extraction of expert's tacit knowledge when constructing a knowledge base of an expert system – in both cases the knowledge is difficult to extract. In chapter 6 we introduced a new paradigm, which facilitates semi-automatic elicitation of knowledge in the form of rules. We successfully applied it to creating a knowledge base of an expert system that recognizes bad bishops in chess middlegames and is able to explain its decisions.

In this chapter, we introduce a novel approach of *Goal-Based Rule Learning* and combine it with the aforementioned paradigm for learning strategic goal-based rules. We harvested the tablebases to extract useful concepts and strategies which the domain expert in close collaboration with the machine learning tool turned into a textbook (and computer aid) for teaching the KBNK endgame. It is important to note that at the beginning of the process the expert was unable to express such precise instructions on his own and was even unaware of some of the important concepts that were later used in the instructions.

7.2 Semi-Automatically Derived Instructions for KBNK endgame

We present here the instructions in the form of goals for delivering checkmate from any given KBNK position. These instructions were semi-automatically derived from the tablebases. In the following hierarchical set of goals, to deliver successfully a checkmate, the chess-player is instructed always to try to execute the highest *achievable* goal listed below. The goals are listed in order of preference, goal 1 being the most preferred. The chess-player is expected to know how to avoid stalemate, piece blunders, and threefold repetitions. Apart from descriptions of the goals we also illustrate most of the concepts behind them.

Goal 1: Deliver Checkmate

A checkmating procedure is the following: two consecutive checks with the minor pieces are delivered, the later one resulting in one of the two types of checkmate positions shown in Fig. 7.1.

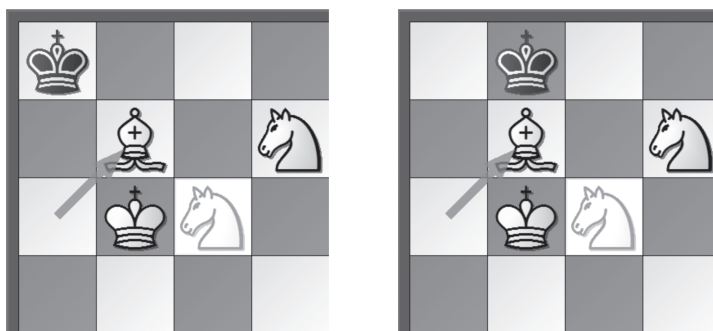


Figure 7.1: Checkmate can be delivered by the bishop or the knight, always in the corner of the bishop's color ("right" corner). Each arrow indicates last bishop's move.

Goal 2: Prepare the Knight for Checkmate

This goal applies when the king and the bishop restrain the defender's king to only two squares: the corner square and a square on the edge of the board right beside the corner square (see Fig. 7.2). The task of the attacker is to prepare the knight so that it is ready for the checkmating procedure.

Goal 3: Restrain Defending King to a Minimal Area Beside the Right Corner

The task of the attacker is to take squares away from the defending king until it is driven to the edge of the board, and consequently to the corner square. The chess-player is advised to aim for the type of position shown in Fig. 7.2 where the king and the bishop restrain the defending king to a minimal area beside the right corner.

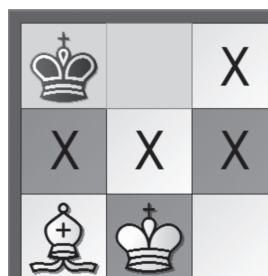


Figure 7.2: A minimal area.

Goal 4: Build a Barrier and Squeeze Defending King

The attacker is advised to build a *barrier* that holds the defending king in an area beside the right-colored corner. When such barrier is built, the attacker should aim to

squeeze the constrained area in order further to restrain the defending king (see Fig. 7.3).

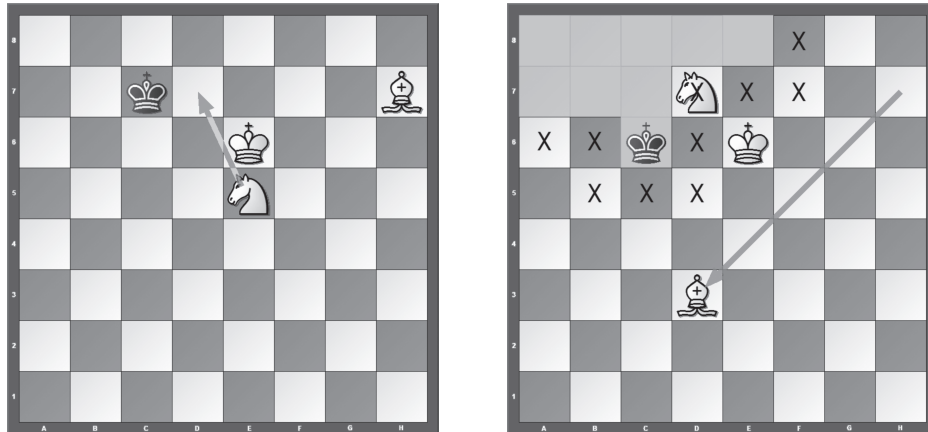


Figure 7.3: In the position shown in the left diagram, the attacking side could build the barrier in the following manner: 1.Ne5-d7 Kc7-c6 2.Bh7-d3!, leading to the position on the right. The area around the right-colored corner to which the defending king is confined, could be squeezed further, *e.g.*, after 2...Kc6-c7 3.Bd3-b5.

Goal 5: Approach Defending King from Central Side

A part of the basic strategy is to drive the opposing king to the edge of the board. In order to achieve this, it is beneficial for the attacking side to occupy squares closer to the center of the chessboard than the defending king does. The attacker should aim to approach the opposing king from the central side of the board.

Goal 6: Block the Way to the Wrong Corner

When the defender's king is already pushed to the edge of the board, the attacker's task is to constrain as much as possible the defending king's way to the wrong-colored corner. At the same time, the attacker should try to keep restraining the king to the edge of the board. Fig. 7.4 shows an example of a typical position that often occurred in simulated games.

Goal 7: Push Defending King Towards the Right Corner

The attacker is advised to push the defending king towards the right-colored corner, at the same time not allowing it to move further away from the edge of the board (see Fig. 7.5).

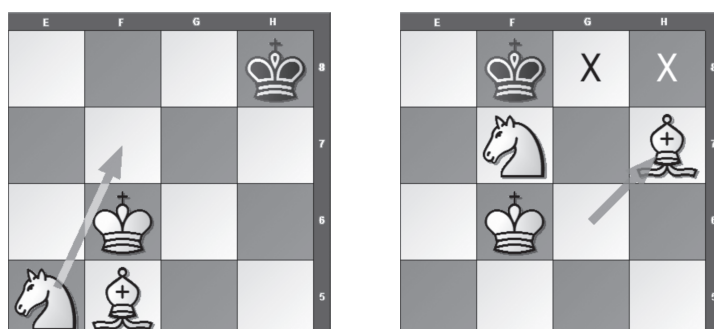
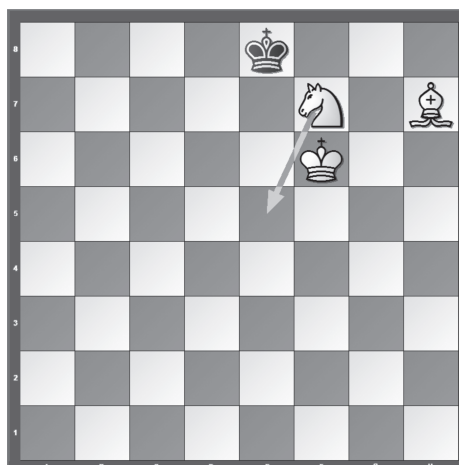


Figure 7.4: In the position on the left, white pieces lure the defending king out of the wrong corner: 1.Ne5-f7+ Kh8-g8 2.Bf5-g6 Kf8 (note that this is the only available square, since h8 is attacked by the knight) 3.Bh7! The last move in this sequence takes under control square g8, and sets up the blockade one square farther from the wrong corner.

Figure 7.5: Black king's distance from the right corner (a8) should decrease, and black king should not be allowed to move away from the edge of the chessboard. This is achieved by the move 1.Nf7-e5, and black cannot resist white's goals: even after suboptimal 1...Ke8-f8 (optimal move according to tablebases is 1...Ke8-d8) white could play 2.Ne5-d7+ Kf8-e8 3.Kf6-e6 and black should move closer to the right corner with the only available move 3...Ke8-d8.



Goal 8: Push Defending King towards the Edge

The attacker is advised to arrange the pieces in such way so that the defending king is pushed towards the edge of the board, and cannot immediately increase the distance from the edge.

Goal 9: Approach with the King

The attacker is advised to move the king closer to the opposing king.

Goal 10: Bring the Knight Closer to the Defending King

The attacker is advised to bring the knight closer to the defending king.

Default Goal: Keep the Kings Close

If none of the above goals is achievable, at least keep the king as close as possible to the defending king, and - if possible - strive for the opposition of the kings.

Table 7.1: The hierarchical model of ordered set of rules.

1. IF *true* THEN *checkmate = true*
2. IF *min_area = true* THEN *knight_prepare = true*
3. IF *min_area = false* THEN *min_area = true*
4. IF *king_area < 70* AND *ndist <= 3*
THEN *king_area should decrease*
AND *king_area minimise*
5. IF *cpdist > 1*
THEN *cpdist should decrease*
AND *king_area should not increase*
AND *mkdist should not increase*
AND *mkdist minimise*
6. IF *edist < 1*
THEN *edist should not increase*
AND *knight_on_edge = false*
AND *wrong_corner_way should decrease*
AND *wk_more_central = true*
AND *wrong_corner_way minimise*
7. IF *kdist < 3* AND *ndist < 6* AND *rcdist > 0*
THEN *edist should not increase*
AND *rcdist should decrease*
8. IF *kdist < 3* AND *edist > 0*
THEN *edist should decrease*
AND *piece_safety = true*
AND *wrong_corner_way should not increase*
9. IF *cpdist > 0* AND *mkdist > 2*
THEN *edist should not increase*
AND *mkdist should decrease*
AND *mkdist minimise*
10. IF *ndist > 2* THEN *ndist should decrease*
11. IF *true*
THEN *kdist should not increase*
AND *mkdist minimise*

7.2.1 The Hierarchical Model of Ordered Set of Rules

The ten goals and the default goal in the presented textbook instructions are based on the hierarchical model of ordered set of rules given in Table 7.1. In Section 7.3, we will describe how the textbook instructions were obtained from these rules. Any of these *goal-based rules* [MGSB09] triggers only if preconditions are true *and* the goal is actually achievable. The description of attributes used in the model is given in Table 7.2

Table 7.2: Descriptions of Attributes

Attribute	Description
checkmate	deliver checkmate within 5 plies
min_area	white king and bishop are on adjacent squares, and constrain black king to only two squares beside the right-colored corner
knight_prepare	white knight is able to attack the square adjacent to the right-colored corner square in just one move
king_area	$8 * \text{the farther diagonal black king can reach from the right-colored corner} + \text{number of squares black king can reach on the farther diagonal}$
wrong_corner_way	$8 * \text{the farther diagonal black king can reach from the right-colored corner} + \text{number of squares black king can reach on the farther diagonal, but only in the direction towards the nearest wrong-colored corner}$
kdist	distance between the kings
mkdist	Manhattan distance between the kings
edist	distance between black king and the edge of the board
ndist	distance between black king and white knight
rcdist	distance between black king and right-colored corner
cpdist	distance between white king and the closest square to black king on a straight line between black king and the very center of the board
piece_safety	white pieces are not closer to black king than to white king
wk_more_central	white king is no closer to the right-colored corner than black king, when $\text{rcdist} < 5$, otherwise always true

7.2. Semi-Automatically Derived Instructions for KBNK endgame

Table 7.3: An example game from the following starting position. White: Kh1, Ba8, Nh7; Black: Kb1 (mate in 31). White played according to the rules in our hierarchical model, Black defended optimally. The suggested goals are given each time the previous goal is accomplished. It is also stated in how many moves (N) the goal is achievable. The moves 7.Bd5+ and 9.Kd3 are suboptimal, each one of them prolongs the checkmating procedure by one move.

Suggestion	N	Resulting play
Approach the black king from the central side.	1	1.Kg1 Kc2
Approach the black king from the central side.	1	2.Kf2 Kd3
Approach the black king from the central side.	1	3.Kf3 Kd4
Approach the black king from the central side.	1	4.Kf4 Kc4
Approach the black king from the central side.	1	5.Ke4 Kb4
Approach with the king.	1	6.Kd4 Kb3
Push the black king towards the edge.	3	7.Bd5+ Kc2 8.Bc4 Kb2 9.Kd3 Ka1
Approach with the king.	1	10.Kc3 Kb1
Bring the knight closer to the black king.	1	11.Ng5 Ka1
Block the way to the wrong corner.	3	12.Ne6 Kb1 13.Nd4 Ka1 14.Nc2+ Kb1
Block the way to the wrong corner.	2	15.Bd5 Kc1 16.Ba2 Kd1
Push the black king towards the right corner.	1	17.Nd4 Ke1
Build a barrier and squeeze black king's area.	3	18.Kd3 Kf2 19.Ne2 Kg2 20.Be6 Kf3
Build a barrier and squeeze black king's area.	2	21.Bc8 Kf2 22.Bg4 Ke1
Build a barrier and squeeze black king's area.	2	23.Nf4 Kf2 24.Nh5 Ke1
Build a barrier and squeeze black king's area.	1	25.Ke3 Kf1
Build a barrier and squeeze black king's area.	3	26.Nf4 Ke1 27.Nd3+ Kf1 28.Kf3 Kg1
Restrain the black king to a minimal area.	3	29.Bh3 Kh2 30.Nf4 Kh1 31.Kg3 Kg1
Deliver checkmate.	2	32.Ne2+ Kh1 33.Bg2#

7.2.2 Generating Example Games

The hierarchical model of rules can be used as a heuristic function for generating example games supplemented by commentary in a form of instructions. An instruction is given each time the previous suggested goal has been accomplished. These games can help the student better to understand the learned strategy. They also illustrate how the teaching process would run with the help of a computer. The student would first read the instructions and then be presented a random position to play against the computer. At any point in the game, the computer is able to give an appropriate suggestion to the student in the form of a goal to accomplish. These suggestions/goals could be further augmented by occasionally displaying a side diagram containing the position associated with the given goal.

In general, a successful execution of a given goal can be either optimal or sub-optimal. For generating example games we chose optimal execution of goals. This does not necessary lead to optimal play, which is not the purpose of instructions for teaching a given endgame anyway. One such example game is shown in Table 7.3; note that an optimal execution of the goals achieves checkmate in two more moves than optimal play.

7.3 The Process of Synthesizing Instructions

In Subsections from 7.3.1 to 7.3.5, we introduce the basics of our approach to goal-based rule learning. In Subsections 7.3.6 and 7.3.7, we describe how the instructions for KBNK were derived semi-automatically from our hierarchical rule-based model, and explain the guidelines for interaction between the machine and the expert in order to obtain a human-understandable rule-based model for playing a chess endgame.

7.3.1 Basic Description of Our Approach

As already mentioned, the rules were induced by a recently developed method for goal-based rule induction [MGSB09]. This method extracts a strategy for solving problems that require search (like chess, checkers etc.). A strategy is an ordered list of goals that lead to the solution of the problem, similar to advice list in Advice Languages [Bra82]. These goals can then be used to teach a human, who is incapable of extensive search, how to act in these domains and be able to solve these problems

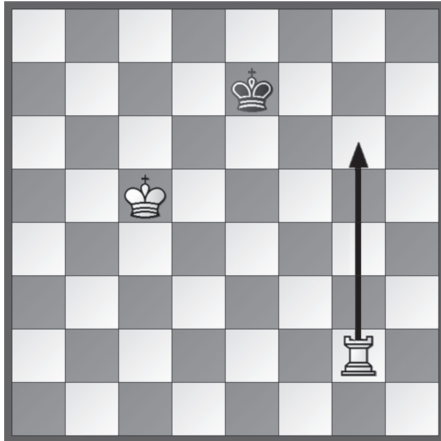


Figure 7.6: Computer: “*What goal would you suggest for white in this position? What are the reasons for this goal to apply in this position?*” The expert used his domain knowledge to produce the following answer: “Black king is close to the edge of the board, but the king is not constrained by white pieces. Therefore I would suggest White to constrain black king.”

simply by following the suggested goals. The method combines ideas from the Argument Based Machine Learning (ABML) [MvB07] with specialized minimax search to extract a strategy for solving problems that require search.

7.3.2 Obtaining Knowledge from Domain Expert

In order to obtain meaningful and human-understandable instructions, the knowledge has to be elicited from a chess expert (in our case this was a FIDE master). Each chess position is described with a set of features that correspond to some well-known chess concepts. The features are obtained by a domain expert as a result of the knowledge elicitation process.

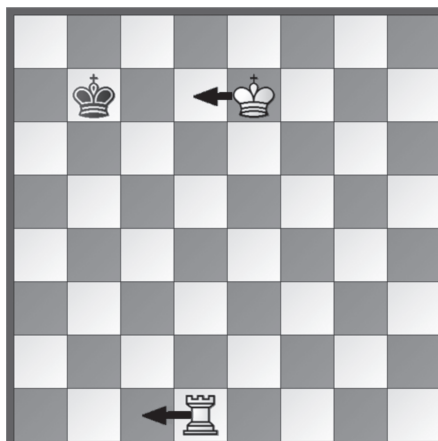
The knowledge elicitation process is similar as in [MGK⁺08; GPB08]: Domain expert and machine learning algorithm improve the model iteratively. We will present a typical interaction between the method and the expert in a simple KKR endgame where we also applied our approach. As a result of this particular step of the interaction (shown in Figures 7.6 and 7.7), a new attribute `king_constrained` was introduced. A demonstration of a similar interaction for the KBNK endgame is presented in Subsection 7.3.7.

7.3.3 Strategic Goal-Based Rules

Our hierarchical model consists of an ordered set of rules of the following form:

IF preconditions THEN goal

Figure 7.7: Computer: “Would you admonish a student if he played 1.Rd1-c1 in this position?” The expert found this move to be perfectly acceptable. Despite of its non-optimality: from the tablebase point of view 1.Ke7-d7 is a much better move - 1.Rd1-c1 (the worst possible execution of the suggested goal) achieves mate in 11 moves whereas after 1.Ke7-d7 only 6 moves are necessary (1...Kb7-b6 2.Rd1-d5!).



The rule’s *preconditions* and *goals* are both expressed by using the aforementioned features. The method used the expert’s argument given in Fig. 7.6 to induce the following rule:

```
IF edist < 3
  AND king_constrained = false
THEN king_constrained = true
  AND edist should not increase
```

where *edist* is the distance between black king and the edge of the board. The subgoal *edist should not increase* was added by the computer. The method recognized that allowing Black to move away from the edge of the board would increase the distance to mate. The expert can accept or reject such suggestions before the rule’s acceptance, but doing so, it is important to rely on his *common knowledge* about the domain.

7.3.4 Allowing Non-Optimal Play

Often, *counter examples* are detected by the method, and presented to the expert. Counter examples are positions where the goal can be achieved, but the resulting play nevertheless leads to *increased* distance to mate. Among such positions, the one with the highest distance to mate is chosen as the key counter example. Figure 7.7 illustrates this idea.

Since human players typically choose a longer path to win by systematically achieving intermediate goals, the expert is instructed to accept counter examples where such subgoals are executed, although they often do not lead to optimal play in

the sense of shortest win against best defence. However, the resulting play in counter examples should lead to overall progress towards achieving the final goal of delivering checkmate. Constraining the black king in the above counter example was judged to lead to such progress.

The expert may also find the execution of the goal in a counter example to be unacceptable. In this case, he may add, modify, and/or remove any of the preconditions and subgoals. Again, doing any of these, it is important that the expert relies on his common knowledge about the domain.

7.3.5 Hierarchy of Goals

When a rule triggers, all the goals higher in the hierarchy are also taken into account. The goal is achievable when at least one of these goals can be executed regardless of the defender's play (optimal or non-optimal). Such hierarchy of goals is typical of a human way of thinking. For example, when the goal is to push the defender's king towards the right-colored corner in the KBNK endgame and the defender resists the goal by allowing the opponent to deliver checkmate (that would not be achievable without the opponent's help), one is expected to see such a possibility. It would be redundant to express goals in the following way: "Push the defending king towards the right corner *or deliver a checkmate, if the opponent plays badly and allows it.*"

7.3.6 Constructing Human-Friendly Instructions from Semi-Automatically Generated Rules

The role of preconditions and non-progressive subgoals is merely to allow a computer program to detect positions where a specific rule triggers and to achieve goals appropriately. All the goals in the instructions are obtained by stating only the progressive subgoal. The exception is the last, default goal, since it is desirable always to be able to give advice to the student. Let us demonstrate this on the following rule (the descriptions of the attributes are given in Table 7.2):

```
IF edist < 1
THEN edist should not increase
    AND knight_on_edge = false
    AND wrong_corner_way should decrease
    AND wk_more_central = true
```

AND *wrong_corner_way minimise*

The precondition $edist < 1$ enables the program to try to achieve the goal only when the defending king is confined to the edge of the board. The progressive sub-goal is *wrong_corner_way should decrease*, so when this rule triggers (*i.e.*, this goal is achievable and no higher rule triggers) the student is given the following advice: “Block the way to the wrong corner.” It is expected from a human to recognize (at least eventually) that moving the knight to the edge, allowing the opponent to move away from the edge, and putting the king closer to the edge than the opponent’s king does not lead to progress. For a computer, such constraints are necessary to enable a sensible execution of the goals.

It is also desirable to obtain sensible diagrams and variations that are supposed to provide a most useful representation of the goals and concepts in a given domain. We obtained these by executing simulations of delivering checkmate from randomly chosen initial positions using the hierarchy of goals. The execution of goals in these simulations was optimal in sense of minimizing the distance to mate (quickest play). For each goal, the position that occurred most frequently in the simulations, was chosen to be presented by a diagram. When several positions occurred equally frequently, more diagrams were used.

7.3.7 Demonstration of Interaction between Computer and Domain Expert in KBNK

The instructions for the bishop and knight checkmate were tailored to students at club level. Our domain expert judged that the skill level of the targeted students should be sufficient for them to be able to calculate chess variations at least three moves (or 6 plies) ahead. This depth of lookahead was therefore set for the depth parameter of our algorithm.

As already mentioned, the expert and the machine learning algorithm improve the model iteratively. A typical interaction in this particular chess endgame is demonstrated in Figures 7.8 and 7.9. Our algorithm is capable of inducing goals that are both achievable and successful in terms of progressing towards delivering checkmate by itself. However, in the critical example in Fig. 7.8 the automatically induced goal (“distance of black king from the edge of the board should decrease”) was characterized by the algorithm as *bad* (*e.g.*, after 1.Bg4-f3 Kg2-h3 the goal is achieved, but

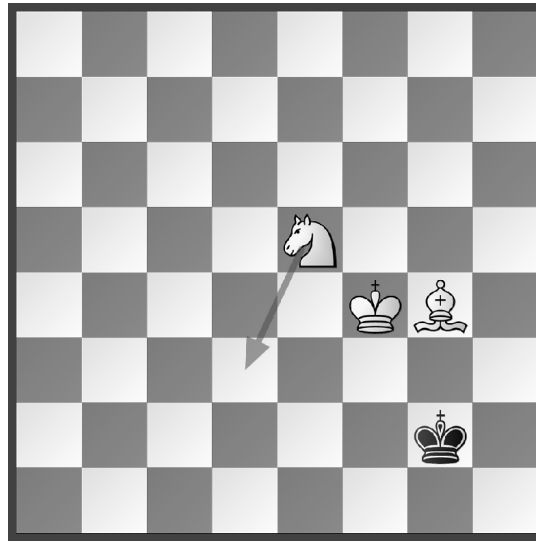


Figure 7.8: Computer: “I suggest the following goal: the distance between black king and the edge of the board should decrease. However, it does not seem to work well in this position. *What goal would you suggest for white in this position? What are the reasons for this goal to apply in this position?*” The expert: “Pushing black king to the edge of the board is fine. However, I find the following goal to be more instructive for the student: *Build a barrier and squeeze the defending king into the corner. Currently such barrier is not yet established.* The move expected from the student is 1.Ne5-d3, achieving the goal.”

distance-to-mate increases).

As an answer to the question posed by the computer, the expert expressed his argument describing the diagrammed position. A new attribute `king_area` was introduced. Note that new attributes also allow the domain expert to introduce important elements of knowledge to be acquired by students. In the present case, the student is advised to build a barrier that holds the defending king in an area beside the corner. When such barrier is built, the student should aim to squeeze the constrained area in order further to restrain the defending king. The comment from the expert was translated into the following argument in computer-understandable form:

```
IF king_area is high
THEN decrease king_area
    AND minimise king_area
```

Important to note here is that “king area” is a new concept introduced by the

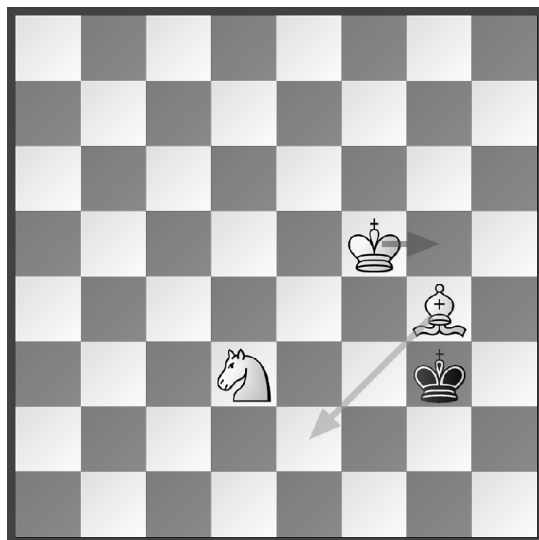


Figure 7.9: Computer: “Would you admonish a student if he played 1.Bg4-e2 in this position?” The expert would not admonish a student, he found this move to be perfectly acceptable. Despite of its non-optimality: from the tablebase point of view 1.Kf5-g5 is a better move - 1.Bg4-e2 (the worst possible execution of the goal of squeezing the area available to the black king) achieves mate in 10 moves whereas after 1.Kf5-g5 only 8 moves are necessary.

expert which he used for explaining his view. The expert is free to use as high-level or low-level concepts as he wishes the targeted students to be presented. In short, this is another way for experts to tailor the level of instructions.

Appropriate translation of the experts’s comment into the language understandable to the algorithm is mandatory. For example, the subgoal *minimise king_area* was included in the argument, as the student has to squeeze the area available to the defending king as much as possible. Without this particular subgoal, another move would be acceptable apart from the one that the expert mentioned: 1.Ne5-c4, which is worse, since it provides more freedom to the defending king.

Among positions where the goal can be achieved, but the resulting play nevertheless leads to an increased distance to mate, the position with the highest distance to mate (see Fig. 7.9) was chosen as the key counter example in this particular interaction.

Allowing non-perfect execution of tasks enables generated instructions to be in a more human-like manner. In chess, human players typically do not play optimally in the sense of shortest win against best defense; they choose a longer path to win

by systematically achieving intermediate goals. The expert would not admonish students if resulting play in counter examples leads to overall progress towards achieving the final goal (delivering checkmate). The way of squeezing the area available to the black king in the counter example in Fig. 7.9, although not optimal in terms of perfect play, was judged to lead to such progress.

7.4 Discussion and Evaluation

The bishop and knight checkmate (KBNK) is regarded as the most difficult of the elementary mates. Several chess books give the general strategy for playing this endgame as follows. Since checkmate can only be forced in the corner of the same color as the squares on which the bishop moves, an opponent will try to stay first in the center of the board, and then retreat in the wrong-colored corner. The checkmating process can be divided into three phases: (1) driving the opposing king to the edge of the board, (2) forcing the king to the appropriate corner, and (3) delivering a checkmate. However, only knowing this basic strategy hardly suffices for anyone to checkmate effectively the opponent.* Another strategy is known as *Delétang's triangles*, involving confining the lone king in a series of three shrinking isosceles right-angled triangles (pioneered by Delétang in 1923 [Del23]). This strategy usually takes five to ten moves longer to deliver checkmate. Since state-of-the-art endgame manuals (*e.g.*, [Dvo08]) prefer teaching the aforementioned three-phase checkmating process, we decided to aim for obtaining the rules for executing that (quicker) strategy.

To the best of our knowledge, no formalized models for KBNK endgame suitable for teaching purposes were derived by any machine-learning programs. As H.J. van den Herik *et al.* in 2002 (and still valid today) nicely put it: “The current state of the art of machine-learning programs is that many ad hoc recipes are produced. Moreover, they are hardly intelligible to human experts. In fact, the database itself is a long list of ad hoc recipes. Hence, the research question is how to combine them into tractable clusters of analogue positions and then to formulate a human-understandable rule.”[vdHUVR02]

Based on the aforementioned Delétang's triangles method, van den Herik constructed a formalized model for playing KBNK endgame and successfully imple-

*For example, grandmaster Epishin (Kempinski-Epishin, Bundesliga 2001) failed to force the defending king to the appropriate corner and the game ended in a draw.

mented it in a chess-playing program [vdH83b]. The knowledge in the model was partitioned into 28 patterned equivalence classes (introduced by Bramer [Bra77]) aimed to correspond to some significant recognizable *features* of the endgame as perceived by chess-players such as “confinement in the wrong corner” and “intermediate class between the large and the middle bishop triangle” (the obtained equivalence classes and patterns are fully described in [vdH83a]). The model was derived from chess theory books, discussions with (grand)masters, and the author’s experience, but without any machine-learning programs or chess-tablebases support. Similarly as with our approach, the resulting knowledge is intended to be used jointly with tree search with a maximum search depth decided in advance and does not necessarily produce optimal play. There are several important differences between [vdH83b] and our approach, the most important two of them being:

- Obtaining the formalized model for KBNK in [vdH83b] required *existence* of some method for playing the endgame in question (in this case, the method discovered by Delétang), while our ABML-based knowledge elicitation process provides a potential for obtaining goal-based instructions for *any* chess endgame where tablebases are available. Note that no known method for delivering checkmate exist for more complex endgames (such as KBBKN, for example).
- Using pattern-classes based model leads to instructions in form of descriptions of *states* the chess-player should aim to achieve from a given position, while our strategic goal-based rules suggest the relative change (improvement) in a position given in terms of *one progressive goal* per instruction. For a student, it seems easier to memorize instructions containing a single progressive goal than a sequence of several states.

In another attempt to obtain a formalized model for KBNK endgame, van den Herik and Herschberg [vdHH86] strived for optimal play, using tablebases. After the very limited success, the task of translating perfect information into a set of rules to be followed by human or computer was reported to be extremely difficult.

The extracted strategy as described in Section 7.2 was presented to three chess teachers (among them a selector of Slovenian women’s squad and a selector of Slovenian youth squad) to evaluate its appropriateness for teaching chess-players. They all agreed on the usefulness of the presented concepts and found the derived strat-

egy suitable for educational purposes. Among the reasons to support this assessment was that the instructions “clearly demonstrate the intermediate subgoals of delivering checkmate.”

We also evaluated the rules by using them as a heuristic function for 6-ply minimax search to play 100 randomly chosen KBNK positions (each requiring at least 28 moves to mate providing optimal play[†]) against perfect defender. We tested two different strategies for cases when the heuristic suggests several moves achieving the goal: either (a) a move that minimizes distance to mate (quickest play), or (b) a move that maximizes distance to mate (slowest play) was chosen. Our rules were able to achieve mate in 100% of the cases using both quickest play (average game length was 32 moves) and slowest play (average game length was 38 moves). Therefore, even with slowest possible realization of strategic goals, the strategic rules are expected to guide a student reliably towards the final goal of achieving checkmate within allowed 50 moves. It is worth noting that state-of-the-art chess engines such as RYBKA 2.1, ZAPPA 1.1, and TOGA II 1.3.1, when limited to a 6-ply search only, were not able to deliver checkmate within 50 moves against an optimal defender from any of the 100 starting KBNK positions.

Our ABML-based approach leads to obtaining domain’s strategic goal-based rules using the same arguments and the same domain language attributes as the expert does. We therefore expect the resulting models to produce “human-like” style of play, in the sense that it would be clearly understandable by human players. As typical for humans, such play would not aspire to minimize distance to win. To verify this claim, we applied our approach to constructing strategic rules for the KRK chess endgame, where it is commonly accepted that a traditional way of delivering mate differentiates from optimal (tablebase) play.

We verified our claim with a kind of a Turing test. Four strong grandmasters were asked to observe 30 games: 10 games played by our KRK chess program guided only by the rules obtained with our ABML-based method, 10 games by a perfect (tablebase) player, and 10 additional games (further to complicate the evaluators’ job), all facing a perfect (tablebase) opponent. They were only told that at least in some of the games white player was a computer program, while black always defended optimally. The grandmasters were asked to express their assessment for each game to what degree (marks 1 to 10) they find the play to be human-like. The

[†]KBNK is a 33-move game in the maximin sense, as it was established after the complete tablebases were computed by Dekker and van den Herik in 1982 [vdH83a].

mark of 1 means that the attacker's play seems totally computer-like, and mark 10 means that it seems totally human-like. The average scores given to our KRK rules by the four grandmasters were 4.1, 7.1, 8.2, and 7.3, while the average scores given to tablebase player were 2.2, 3.1, 1.8, and 2.0, an obvious difference!

7.5 Final Remarks

We developed a procedure for semi-automatic synthesis of textbook instructions for teaching the KBNK endgame, accompanied by example games containing generated instructions. The presentation of the derived strategy includes, importantly, a number of concepts and key positions from this endgame that help the human learner to easily understand the main principles of this strategy. The key positions were detected automatically from simulated games played by the derived strategy. The key positions in Figures 7.3, 7.4, and 7.5 belong to a frequent sequence of seven moves in tablebase play. In our case, this sequence was reduced to the three key positions and conceptualized in terms of goals along the sequence. In contrast to memorizing the optimal sequence itself, the extracted generalization also enables correct play against suboptimal defence.

The derived strategy is human-friendly in the sense of being easy to memorize, but produces suboptimal play. In the opinion of chess coaches who commented on the derived strategy, the tutorial presentation of this strategy is appropriate for teaching chess students to play this ending. We view the positive assessment of derived textbook instructions by chess coaches as a confirmation that our approach is able to facilitate knowledge extraction from the tablebases.

We also explained the guidelines for the interaction between the machine and the expert in order to obtain a human-understandable rule-based model for playing a chess endgame, and how the instructions, including illustrative diagrams, could be derived semi-automatically from such a model.

There are at least two obvious directions for future work. First, to create a computer tool for teaching the KBNK endgame. All the main ingredients are already available and all that remains is to package them into an actual application as described in Subsection 7.2.2. The second possible line of future work is to use the described procedures to synthesize instructions for much harder endgame, namely KBBKN.

Part III

On The Nature of Heuristic Search in Computer Game Playing

Chapter 8

Monotonicity Property of Heuristic Evaluation Functions

The third part of the dissertation aims at improving the understanding of properties of heuristic search in both human and computer problem solving, in particular with respect to computer game playing.

In this chapter, we analyze the properties of successful evaluation functions in game playing. We demonstrate that backed-up values of the nodes in the search space have to tend to monotonically approach to the terminal values of the problem state space with the depth of search. A theoretical model about a *monotonicity property* of heuristic evaluation functions is designed. We show empirically that the evaluation functions of typical chess programs tend to have this property that enables the program to play with a sense of direction towards a desirable goal.

Some of possible impacts of the monotonicity property of heuristic evaluation functions on the theory and practice of game playing are discussed. Among other findings, we arrive experimentally at the following claims.

- Heuristic evaluation functions do *not* approximate some “ideal” or “true” values (as it is commonly assumed).
- Heuristic evaluations obtained by a search to different search depths are *not* directly comparable among each other.
- Backed-up heuristic values should *not* be invariant with deeper search, as game-theoretical values in the theoretical minimax model are.

- The same backed-up heuristic values obtained by different evaluation functions do *not* necessarily reflect the probability of winning in the same way.

We also show that taking into account the monotonicity property of heuristic evaluation functions can provide a heuristic-search based program with an ability to solve a difficult type of problems: detecting fortresses in chess.

8.1 Heuristic Evaluation in Game Playing

The minimax principle is applied in game playing as follows. The rules of the game define the outcomes of the game in the terminal positions of the game, that is, in the leaves of the game tree. These are called *game-theoretic values* of positions. If the game ends in such a terminal position, this value is the final result of the game, that is, this value is the score obtained by the players. The rules of the game do not define the values of non-terminal positions. These values can be determined by back-propagation from terminal positions and applying the *minimax rule*. So the value of any non-terminal position P is equal to the score of the terminal position of the game that starts in P and is played optimally by both players. An optimal play (principal variation) from position P is, accordingly, a sequence of moves that do not change the value of the positions along this sequence. So the property of the principal variation is that it preserves the value of the game. The so-obtained values of the nodes throughout such completely generated game tree therefore respect the minimax relation.

The minimax principle, often implemented by the alpha-beta algorithm, is widely used as the basis for computer game playing. The above theoretical construction is applied in practice with the modification that in games of any interest, it is computationally prohibitive to search as far as terminal positions. Instead, search has to stop short of reaching terminal positions, typically when some depth limit is reached and quiescence criterion satisfied. The part of the game tree that is generated for search is called a *search tree*, and a heuristic evaluation function is applied to terminal positions of the search tree. These terminal positions are evaluated “statically” as if they were terminal positions of the game. Of course, they cannot be evaluated by the rules of the game, so they are evaluated heuristically. The heuristic evaluations of non-terminal positions of the search tree are obtained by the minimax back-up

rule. Therefore, the values of the nodes in the search tree also respect the minimax relation.

In Subsection 8.1.1, we discuss what may be “ideal” heuristic values. In Subsection 8.1.2, we explain why heuristic values should enable direction-oriented play. Finally, in Subsection 8.1.3, we present our theoretical model about the *monotonicity property* of heuristic evaluation functions.

8.1.1 What are “Ideal” Heuristic Values?

It is assumed that heuristic evaluation functions approximate some “ideal” or “true” values, and that such values could be assigned to any non-terminal position in the game tree. Since these “true” values are not known, it is accepted that they have to be approximated heuristically. For example, Luštrek *et al.* give conditions under which real-valued evaluation functions outperform discrete-valued evaluation functions, proposing a model that uses real numbers for “both *true* and heuristic values” [LGB05]. In the proposed model, static heuristic values are obtained “by corrupting the true values at depth d with error representing the fallibility of the heuristic evaluation function”. As another example we could take a typical view among chess players: they commonly believe that evaluations given by chess programs try to approximate some “true” values of positions that are a subject of computer analysis (see Figure 8.1).

However, what are these “true” values? This question has been largely ignored. Clearly, they are not backed-up values from the game-theoretic values at the leaves of the game tree using minimax, since it is well known that game-theoretic heuristic values alone would not produce desirable results ([SK98], see Section 8.1.2). Such “true” values would not be useful for practical game playing, therefore aiming to approximate them would be useless. Heuristic evaluations are typically multivalued and they are supposed to reflect a goodness* of a particular position. It is well known that searching deeper generally leads to stronger play. A common belief is that searching deeper leads to better approximations of the value of the root node of the search tree (after minimaxing) to the unknown “true” value of the position at the very same root node. That is, it is typically assumed that searching deeper results in more accurate evaluation in terms of approaching to the unknown “true” value of

*Actually, what exactly this value means was never strictly defined. Various authors viewed this value as position’s “worth”, “merit”, “strength”, “quality”, or “promise” [Abr89].

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS



Program	Evaluation
CHESSMASTER 10	0.15
CRAFTY 19.19	0.20
CRAFTY 20.14	0.08
DEEP SHREDDER 10	-0.35
DEEP SHREDDER 11	0.00
FRITZ 6	-0.19
FRITZ 11	0.07
RYBKA 2.2n2	-0.01
RYBKA 3	-0.26
ZAPPA 1.1	0.13

Figure 8.1: Lasker-Capablanca, St. Petersburg 1914, position after white's 12th move. The table on the right shows backed-up heuristic evaluations obtained by various chess programs, when evaluating the diagrammed chess position using 12-ply search. Chess programs usually use heuristic evaluations where advantage of one unit represents material advantage of one pawn (or equivalent by means of accumulated pluses of positional features). Chess players are nowadays used to computer evaluations and even widely accepted the *centipawn* as the unit of measure used in chess as measure of the advantage, a centipawn being equal to 1/100 of a pawn. As the figure clearly shows, different programs typically assign different evaluations to a given position, even when using the same depth of search. This leads to misleading impression that the programs try to approximate some unknown “true” heuristic value of the position being evaluated (which is obviously not meant to be the same as the game-theoretic value, which could be either “win”, “draw” or “loss”). However, as we intend to demonstrate, this conclusion is false: such “true” values cannot exist.

the root-node position, and that the “true” value should *not* change with increasing search depth. Consequently, this assumes that a “perfect” heuristic evaluation function would statically (*i.e.*, without any search) assign the “true” value to the position in question and that searching deeper would not affect this evaluation. This corresponds to the assumption that the “true” heuristic values throughout the tree respect the minimax relation, and that heuristic functions should therefore also aim to respect the minimax relation. However, we will show that this generally accepted view does not correspond to what actually happens in computer game playing.

When giving arguments in support of look-ahead, Pearl explains the notion of *visibility*, which says that since the fate of the game is more apparent near its end, nodes at deeper levels of the game-tree will be more accurately evaluated and choices based on such evaluations should be more reliable [Pea83]. We will demonstrate that the

improved accuracy of static heuristic evaluations at higher depths (and consequently the accuracy of the backed-up evaluations at the root of the search tree) does *not* lead to assigning some better approximations to unknown “true” or “ideal” heuristic values to the position at the root of the search tree after minimaxing, but that the value of the root position *should* be changing with increasing search depth.

8.1.2 Direction Oriented Play

Superficially it might seem that the “true” values have a simple definition - that they are simply minimax backed-up values from the game-theoretic values at the leaves of the game tree. However, such definition does not produce “true” values *useful* for practical game playing. To emphasize the difference between game-theoretic values and “ideal heuristic values”, from now on we will be referring to these “ideal heuristic values” as *utility values* or *utilities*. A function that assigns the utility to a position will be called a utility function.

The purpose of utility function is to guide the game-tree search. Utilities have to enable a program to find a *direction* of play towards a win, not only to maintain a won position. In addition to reflecting the game-theoretic value of a position, utility values should in some way also reflect the progress towards the end of the game. For chess, for example, it is well known that the game theoretic value could not be a useful utility function. Scheucher and Kaindl advocate that a heuristic evaluation function should be multivalued to be effective and that game-theoretic heuristic values alone would not produce desirable results [SK98]. In chess there are only three possible outcomes: win, draw and loss. Given a won position, this utility function would just ensure that the value “win” is maintained, without any guarantee of eventually winning, since such a program would not be able to discriminate between a position with a slight advantage from one that is clearly won (see Fig. 8.2).

This actually happens when one of the state-of-the-art chess programs is confronted with the simple task of winning the king and rook versus lonely king endgame without the use of chess tablebases,[†] and is limited with sufficiently low search depth. It turns out that the program behaves in the following way:[‡] when using a search depth of 4 and higher, it assigns the same heuristic evaluations to all winning po-

[†]Chess tablebases, indicating best moves for every position, exist for chess endgames up to 6 pieces (including the kings) and are commonly used by chess programs.

[‡]The program where this phenomenon occurs is “RYBKA 2.1c 32-bit”. In the year 2006, this was the highest rated chess program. The phenomenon no longer occurs in the later versions.

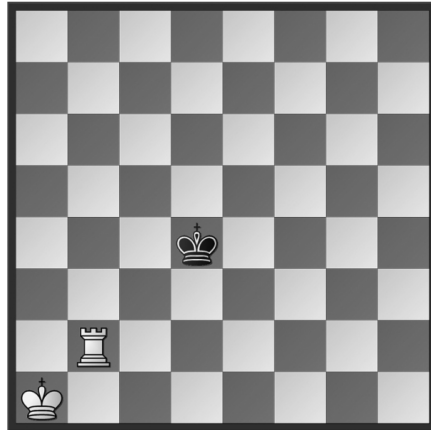


Figure 8.2: In the simple chess endgame shown in the figure, the extra rook guarantees the white player an easy win. By moving his pieces in appropriate manner white will force the black king on the edge of the chessboard and eventually deliver a mate. All terminal nodes of the game tree from this position on are wins, except the ones where white had lost the rook and the ones that end in a stalemate. If utility function with game theoretic values was used for playing this endgame, white pieces would just wander around the board, only maintaining a won position and would never (except by pure luck) deliver mate in time (limited by the 50 moves rule), providing a sufficiently low search depth. However, state-of-the-art evaluation functions enable chess programs to reliably deliver checkmate within prescribed 50 moves even at the shallowest depths of search.

sitions in this endgame (concretely, it is the numerical value of 4.92), regardless of search depth. That is, although the program's evaluation function assesses positions in this endgame as won, it fails to distinguish between non-equally promising positions for achieving the final goal: delivering checkmate. This results in rather ridiculous play by the winning side. In simulations from 100 randomly chosen mate-in-16 positions, where the program played against the black player defending optimally (using tablebases), the program did not manage to deliver checkmate within prescribed 50 moves in several games, even at 12-ply search. Despite the fact that the program is aware of the 50-move rule,[§] it does not help it to always avoid the draw, when depth of search is limited. The same program, when using the shallow search of only 2 plies (*i.e.*, when the phenomenon does not occur) checkmates the opponent in 100% of the games played from the same randomly chosen positions, also finishing

[§]The basic rules of the game according to FIDE say: "The game may be drawn if each player has made at least the last 50 consecutive moves without the movement of any pawn and without any capture."

the task in considerably less moves on average. It is worth noting that the backed-up evaluations of the 2-ply search were on average monotonically increasing during the simulated games, while at search depths where the phenomenon occurs they always stayed the same, unless the depth of search sufficed to find a principal variation that ends in checkmate (when the heuristic evaluation is no longer necessary). Note also that an evaluation function of a program that assigns the same heuristic values to all winning positions behaves as if the minimax relation between heuristic evaluations in a game tree should be respected.

Another simple definition of the “true” or “ideal” heuristic values may seem to be a distance-to-win metric. That is, it may seem that the length of the principal variation from the given position up to the leaves of the game tree that represent theoretical wins could be a successful utility value. However, an utility function that would use such utilities (that are obviously not invariant along the game tree) would not be able to distinguish between theoretically drawn positions with different probability of actually achieving a draw in a game between fallible opponents. We will demonstrate, however, that evaluation functions in typical chess programs tend to have this desirable property that enable the program to distinguish between differently promising positions in games between fallible players.

8.1.3 Our Theoretical Model

Direction oriented play (as opposed to advantage-preserving play) is a property of every successful program in all typical games where heuristic search is used, therefore it seems reasonable to expect this property to be reflected somehow in the programs’ heuristic evaluations.

The theoretical model in Figure 8.3 is designed for games where three game-theoretical results are possible (as in chess and checkers, among several other two-player games) and shows our expectations. With increasing search depth, backed-up evaluations of won positions (in theoretical sense: white wins providing optimal play by both sides) will on average be increasing, evaluations of lost positions will be decreasing, while evaluations of positions with game-theoretical value “draw” will be converging towards 0 and search will eventually end in terminal nodes that represent theoretical draw.

We investigate experimentally this property of the evaluation function of the well-known open source chess program CRAFTY as a typical representative of chess eval-

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS

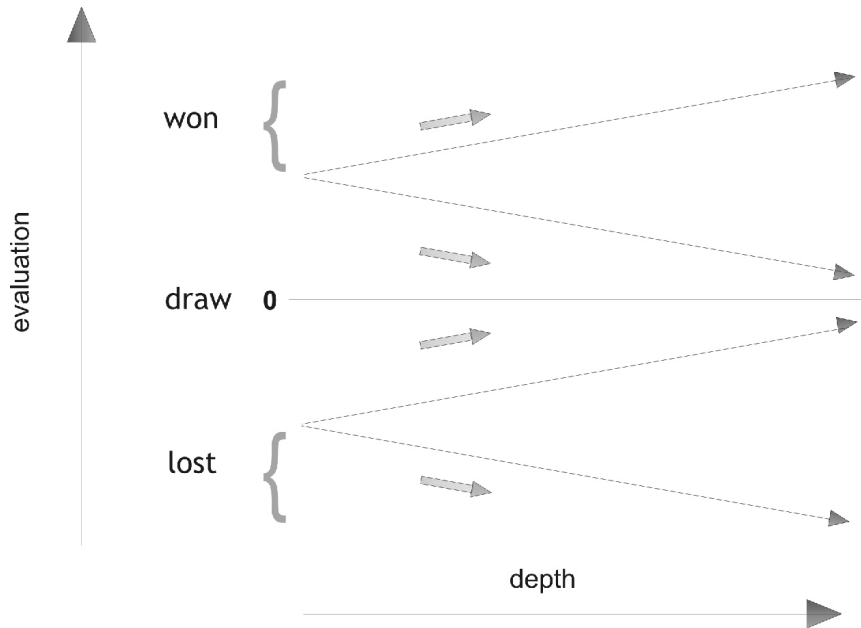


Figure 8.3: Theoretical model about the monotonicity property of heuristic evaluation functions.

uation functions. The results, actually confirmed by currently one of the strongest chess programs, RYBKA, indicate that evaluation functions of strong chess programs have this desirable property that gives the program's play a sense of direction.

Our claim is the following: Backed-up values of the nodes in the search space have to tend to approach monotonically to the terminal values of the problem state space. Consequently, backed-up heuristic values at the root of the search tree should not be invariant with deeper search and could not behave as approximations to some "ideal" or "true" heuristic values. Existence of such "ideal" or "true" heuristic values that could possibly be assigned statically to any node in the problem state space implies that the minimax relation between them is satisfied. However, then the evaluation function would not be able to enable the program to play with a sense of direction towards a win. With other words, heuristic evaluations cannot behave as the terminal position values in the theoretical minimax model. Our theoretical model also implies that heuristic evaluations obtained by searching to different search depths are not directly comparable, in contrast to what is generally assumed both in literature and in practical applications.

Regarding the related work, we believe that there has been no study about the

implications of the monotonicity property of heuristic evaluation functions. Luštrek *et al.* highlight that multiple values make it possible to maintain a direction of play towards the final goal.[¶] [LGB05] Gomboc *et al.* show that it suffices for evaluation functions to tell only whether a certain position *is* better than some other position, and not *how much* better. [GMB03] Note that their demonstration why ordinal correlation is relevant to heuristic search is in line with our claim that evaluation functions do not necessarily need to (and actually do not) express exact goodness of a certain position in order to be successful. Donkers *et al.* examine three types of evaluation functions: *predictive*, *probability estimating*, and *profitability estimating* evaluation functions. They also investigate how evaluation functions can be compared to each other. [DvdHU05] Note that each of the three types of evaluation functions enables direction oriented (as opposed to advantage-preserving) play. Several authors have studied properties of heuristic evaluation functions, particularly in respect of the propagation of static heuristic errors through minimaxing.^{||} Various authors conducted *go-deep* experiments [Hei98; Hei99a; Hei99b; Hei01; Hei03; HN97; JSB⁺97; Ste05], as we did in the present study. All this related work does not mention the monotonicity property of heuristic evaluation functions.

To the best of our knowledge, Newborn ([New75]; pp. 84-86) was the only one to analyze the monotonicity property of heuristic evaluation functions, back in the year 1975, when describing the contest between COKO III and GENIE in Chicago, 1971. Nevertheless, the monotonicity property of heuristic evaluation functions remained neglected both in the literature and in computer chess programming. The aim of our work presented in this chapter is not only to provide extensive analysis of this property, but also to show its implications for both the theory and practice of game playing.

[¶]They also add: “Given that multivalued position values are necessary for playing well, it is natural to also use multivalued true values. These values are true in the sense that they guide a program to play optimally.” This statement further supports our thesis about common belief that the programs are trying to approximate some unknown “ideal” or “true” heuristic values by searching deeper and back-propagating obtained heuristic evaluations by the rules of minimax.

^{||}An interested reader could find an overview on minimax pathology in doctoral dissertation of Aleksander Sadikov [Sad05].

8.2 Experimental Design

The chess program CRAFTY was used as a typical representative of computer chess programs. To verify that the results obtained with CRAFTY are likely to be generalized to other chess programs, we conducted a control experiment with RYBKA. The programs were used to analyze more than 40,000 positions from real games from World Chess Championship matches in a *go-deep* way: Each position occurring in these games after move 12 was searched to a fixed depth ranging from 2 to 12 plies. A smaller set of 8,000 positions was searched up to 14 plies using CRAFTY. Search to depth d means d ply search extended with *quiescence search* to ensure stable static evaluations.

To determine the best available approximation of the utility of each analyzed position, the backed-up evaluation at the deepest search depth served as an “oracle”. We devised six different groups of positions based on their estimated utility values, as given in Table 8.1. Evaluations by computer chess programs are given by the following standard: the more positive evaluations mean the better position for white and the more negative evaluations mean the better position for black, while evaluations around zero indicate an approximately equal position. In usual terms of chess players, the positions of Groups 1 and 6 could be labeled as positions with “decisive advantage”, positions of Groups 2 and 5 with “large advantage”, while Groups 3 and 4 consist of positions regarded as approximately equal or with a “small advantage” at most.**

Table 8.1: Number of positions in each of the six groups of data in three data sets. The groups were devised based on backed-up heuristic evaluation values obtained at search depth of 12 plies (CRAFTY₁₂, RYBKA₁₂) and 14 plies (CRAFTY₁₄).

Group	1	2	3	4	5	6
Evaluation (x)	$x < -2$	$-2 \leq x < -1$	$-1 \leq x < 0$	$0 \leq x < 1$	$1 \leq x < 2$	$x \geq 2$
CRAFTY ₁₂	4,011	3,571	10,169	18,038	6,008	6,203
CRAFTY ₁₄	422	332	1,875	3,570	1,081	783
RYBKA ₁₂	1,263	1,469	9,808	22,644	3,152	2,133

For each data set and for each group separately we observed behavior of the backed-up evaluations with increasing depth of search.

**Of course, this is only an approximation: The terms “decisive advantage”, “large advantage”, and “small advantage” are not strictly defined in the literature.

8.3 Experimental Results

The comparison of backed-up evaluations obtained at adjacent search depths shows different behavior for positions of each group of our test data. The graph in Figure 8.4 clearly shows that backed-up heuristic evaluations for Groups 1 and 6, where positions are likely to be within the zones of theoretical win and loss in our theoretical model, on average monotonically increase with increasing search depth in positions with a decisive advantage for the white player (*won* positions), and monotonically decrease with increasing search depth in positions with a decisive advantage for the black player (*lost* positions from the perspective of white player).

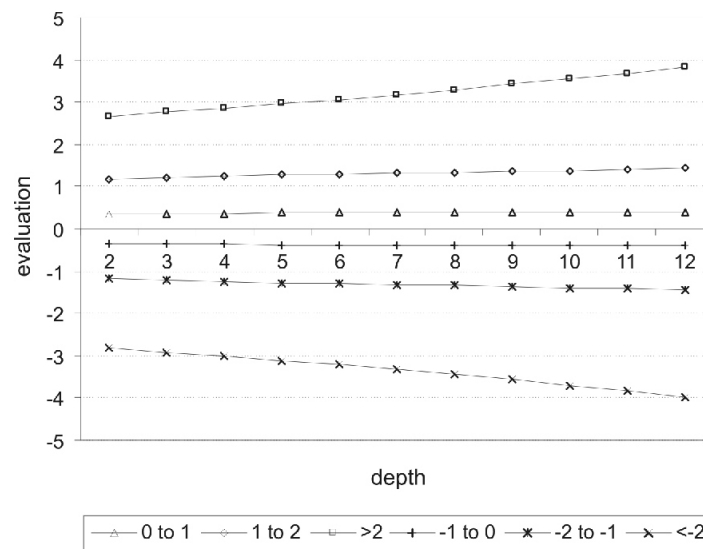


Figure 8.4: Average backed-up evaluations at different search depths for each group of the test data.

As the graph in Figure 8.5 demonstrates, the changes between backed-up evaluations belonging to adjacent search depths are significantly higher for the positions with a decisive advantage than for positions of the other groups.

Figure 8.6 demonstrates that the evaluations gradually approach those at the deepest search depth. Comparison of backed-up evaluations obtained at different search depths to the backed-up evaluation at the highest search depth shows that the former tend to be pessimistic in actually won positions, while in lost positions they tend to be optimistic.

In the graphs of Figures 8.7 and 8.8, each curve represents the distribution of av-

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS

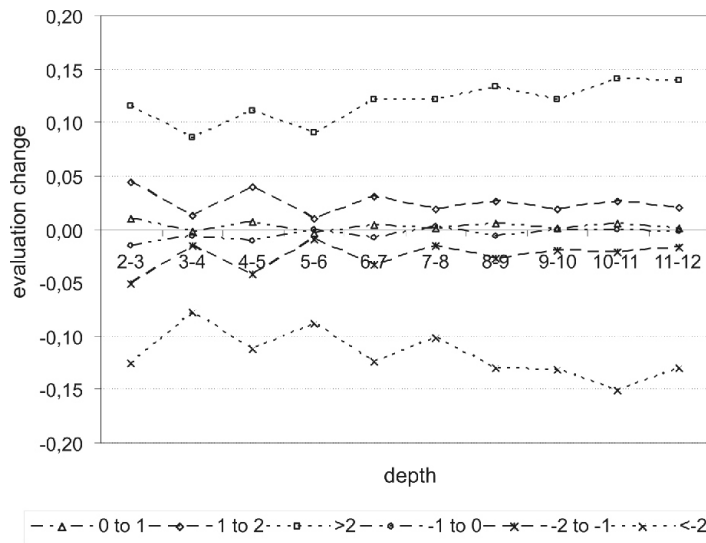


Figure 8.5: Changes between backed-up evaluations belonging to adjacent search depths.

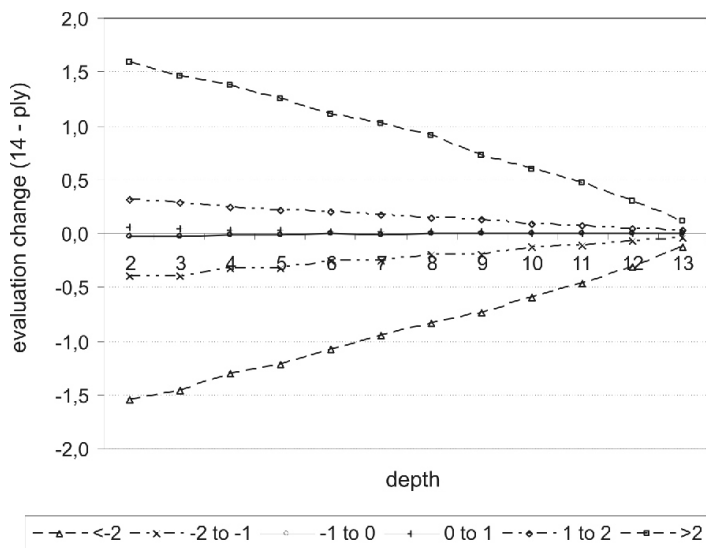


Figure 8.6: Average difference between backed-up evaluations resulting from searches to corresponding depths and the backed-up evaluations at the highest search depth.

average deviations of backed-up evaluations at a given search depth from 14-ply search backed-up evaluations. We considered small intervals of 0.05 difference in backed-up evaluation (both in positive and negative direction from 0, which corresponds to the backed-up evaluation obtained at the highest search depth). For evaluations at

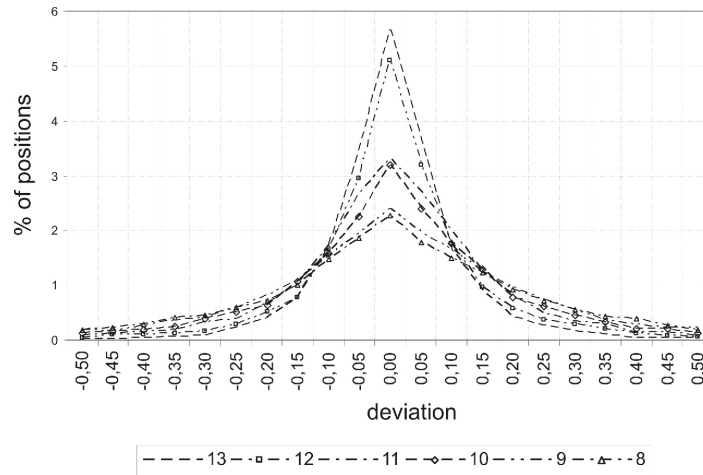


Figure 8.7: Distribution of deviations of backed-up evaluations at different search depths from the backed-up evaluation obtained at 14-ply in balanced positions of Groups 3 and 4.

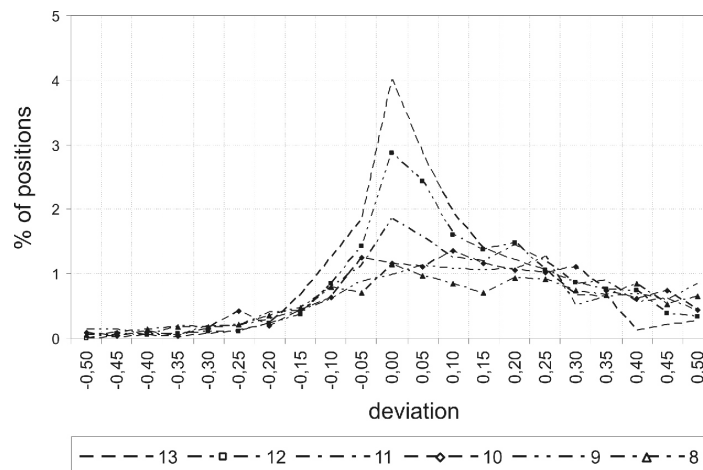


Figure 8.8: Distribution of deviations of backed-up evaluations at different search depths from the backed-up evaluation obtained at 14-ply in won positions of Group 6.

each depth, we checked what percentage of them falls in a given interval.

The result is a graph showing the distribution of deviations. Each curve represents such a distribution for a given search depth. In both such graphs, positive deviation means that in a given interval the average backed-up evaluation at particular search depth was lower than the one obtained from the highest depth of search. Given

that backed-up evaluations gradually approach those at the highest search depth (see Figure 8.6), the deviations for greater search depths should be (and actually are) distributed closer to zero.

Symmetry of such graph with respect to zero deviation means that both positive and negative deviations from backed-up evaluations at the highest search depth are equally represented. In accordance with our claim that heuristic evaluations tend to increase with increasing search depth in won positions, the graph is approximately symmetrical only for more or less balanced positions of Groups 3 and 4 (see Figure 8.7). In positions with a decisive advantage the described bias of evaluations results in non-symmetrical graphs. In the graph of Figure 8.8, only positions of Group 6 were taken into account. The obvious inclination to the right means that in prevalent part of won positions, the evaluation at the highest search depth was higher than the evaluations at shallower depths. Similar inclination (but in the opposite direction) was noticed in the equivalent graph for lost positions of Group 1.

The presented results demonstrate that in won (lost) positions, CRAFTY's backed-up evaluations as a result of search to different search depths tend to increase (decrease) monotonically with increasing search depth. But with what confidence can we expect the backed-up heuristic values obtained from deeper searches to be higher (or lower) due to the monotonicity property of the program's evaluation function? Does this phenomenon occur on regular basis, or only on average? Do the backed-up heuristic values increase (decrease) more rapidly with increasing search depth monotonically with the utility value in won (lost) positions?

In order to answer these questions, we further divided the data of won positions of Group 6 into four subsets, based on backed-up heuristic evaluation values obtained at search depth of 12 (similarly as earlier, the highest search depth served as the best available approximation of the utility value of each analyzed position, see Section 8.2). For each subset separately we observed:

1. the rates of the backed-up evaluation at the highest search depth being higher than the backed-up evaluation at each particular depth, and
2. the average backed-up evaluations at each depth.

Figure 8.9 shows that the confidence of expectation of the backed-up heuristic values obtained from deeper searches being higher in won positions depends on the utility value of a position. Moreover, the graph shows that it is more likely that the

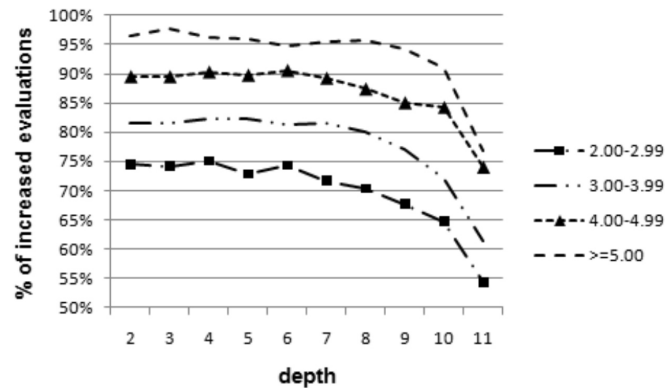


Figure 8.9: The rates of the backed-up evaluation at the highest search depth being higher than the backed-up evaluation at particular depth for each subset of won positions of Group 6, obtained with CRAFTY).

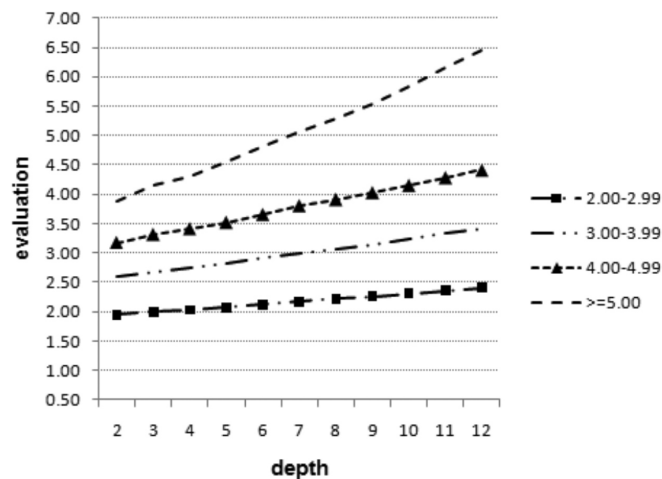


Figure 8.10: The average backed-up evaluations at each depth for each subset of won positions of Group 6, obtained with CRAFTY.

backed-up evaluation will increase when the difference between the depths of search are bigger.

Figure 8.10 shows that the backed-up heuristic values in won positions indeed increase more rapidly with increasing search depth monotonically with the utility value of the position.

We repeated the experiment with the program RYBKA to check whether our re-

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS

sults with CRAFTY are likely to hold for other chess programs. Figures 8.11 and 8.12 (compare to Fig. 8.4 and Fig. 8.10, respectively) show the corresponding results with RYBKA, which confirm that the behavior of CRAFTY's and RYBKA's heuristic evaluation functions reflect the monotonicity property in a similar way.

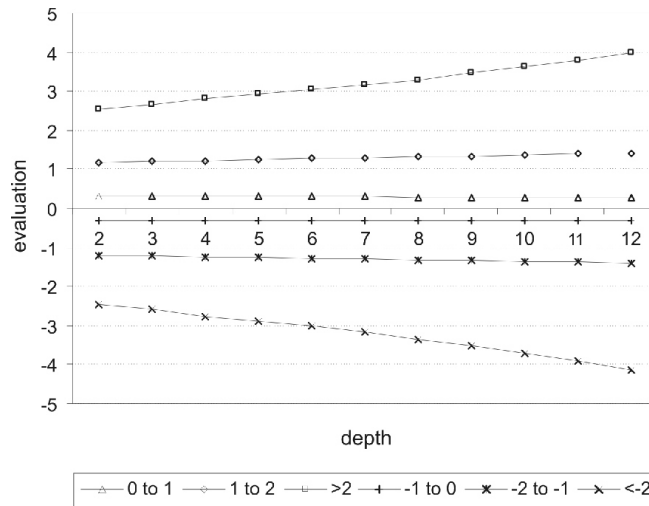


Figure 8.11: Backed-up evaluations depending on search depth obtained with RYBKA (compare to Fig. 8.4).

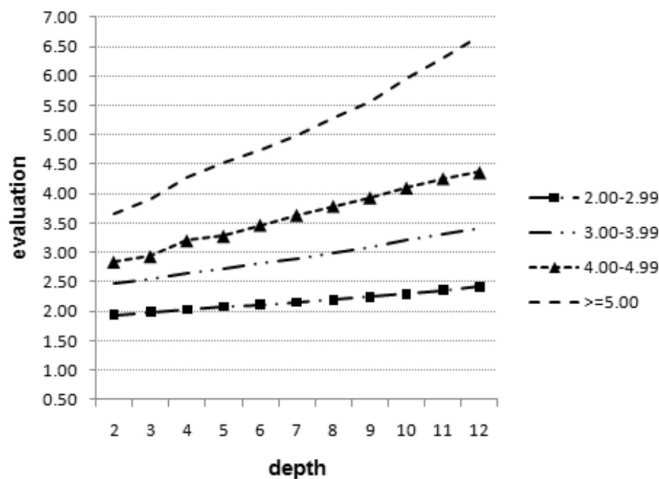


Figure 8.12: The average backed-up evaluations at each depth for each subset of won positions of Group 6, obtained with RYBKA (compare to Fig. 8.10).

So far the results did not seem to confirm another prediction of our theoretical model: that the backed-up evaluations of positions with game-theoretical value “draw” will be converging towards 0 and search will eventually end in terminal nodes that represent theoretical draw. Was the chosen interval of utility values that are supposed to reflect theoretically drawn positions (*i.e.*, drawn positions provided optimal play by both sides) too big? Or is search up to 12 plies too shallow?

In order to answer these questions, we now divided the data of presumably drawn positions of Groups 3 and 4 into several subsets, again based on backed-up heuristic evaluation values obtained at search depth of 12. Figures 8.11 and 8.12 show the obtained results with the chess program RYBKA. It turned out that backed-up evaluations of RYBKA are more closely distributed around the value of 0 than CRAFTY’s (we will return to this observation in Section 8.4). Using RYBKA in the following experiments therefore allowed us to divide the data into smaller, but still well represented subsets.

In Fig. 8.13, the chosen interval of backed-up evaluations obtained at the highest search depth, which served to divide the data into subsets, was 0.10. The average number of positions in the subsets was 1,600 (the minimum number being 436, and the maximum number being 3,618). The value of 0 was treated separately and was assigned to a special interval, represented by 2,053 positions.

The results show that backed-up evaluations on average indeed monotonically approach to the value of 0, however, only in those intervals where the approximated utility value is sufficiently close to 0. As it could be seen from Fig. 8.13, it is when the backed-up evaluations obtained as a result of 12-ply search are within the interval $[-0.50, 0.50]$ approximately. According to our theoretical model and provided that RYBKA’s evaluation function is a successful one, positions of these subsets are more likely to be within theoretical draw. Positions where the backed-up evaluations obtained as a result of 12-ply search are outside this interval are less likely to be drawn provided optimal play. All the curves are ordered according to the value at search depth of 12 in the interval they represent, note also that none of the curves cross each other.

In Fig. 8.14, the data was divided in a similar way, only the chosen interval of backed-up evaluations obtained at the highest search depth was lowered to 0.03. The average number of positions in the subsets was 915 (the minimum number being 474, and the maximum number being 2,053). The results show behavior of RYBKA’s evaluation function, when the approximated utility values are closer to 0. From this

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS

figure it is even more clearly visible that the backed-up evaluations that are likely to be theoretically drawn monotonically approach towards the value of 0 with increasing depth of search.

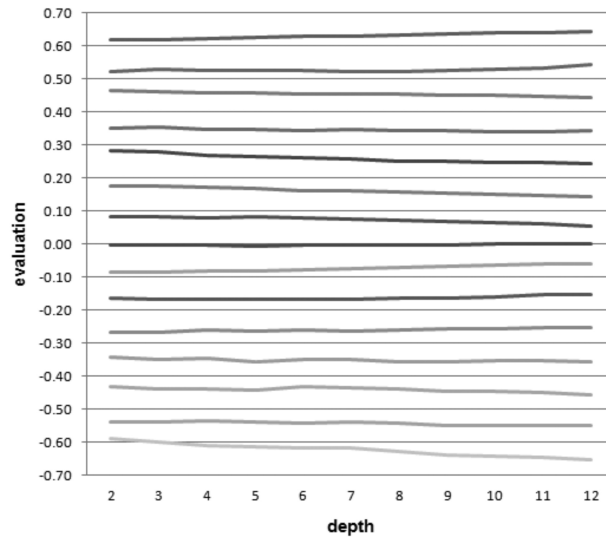


Figure 8.13: Backed-up evaluations depending on search depth for different subsets of approximately equal positions, obtained with RYBKA (size of the interval: 0.10).

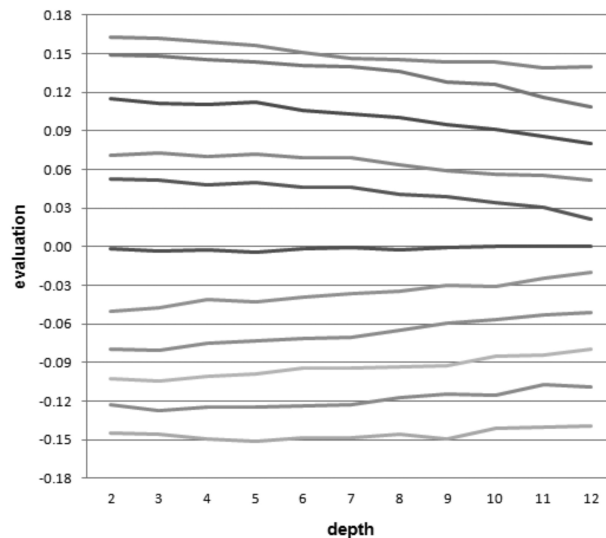


Figure 8.14: Backed-up evaluations depending on search depth for different subsets of approximately equal positions, obtained with RYBKA (size of the interval: 0.03).

We should point out that results obtained on the data set of positions that were the subject of searching up to 14 plies lead to the same conclusions. Because the data set of positions that were searched up to 12 plies is much larger (see Section 8.2), we chose to present the results obtained on this data set on most occasions (except in Figures 8.7 and 8.8). In the large data set, the subsets that were obtained by further divisions of the original data were represented with at least several hundreds of positions.

8.4 Heuristic Evaluation Functions and Probability of Winning

In this section, we will show that backed-up heuristic evaluations tend to reflect the probability of winning in a game between two fallible players. We will also demonstrate that the same backed-up evaluation values obtained by different evaluation functions do not reflect the probability of winning in the same way.

Up to now it was not important who were the players in the games we chose for analysis with chess programs CRAFTY and RYBKA, and what were the game scores - heuristic evaluation function of a particular program should of course assign the same values to the same positions, regardless of this information. Now, however, we will observe what was the game result (win, draw, or loss) that was obtained from a particular position that was the subject of computer analysis. For the experiments in this section, exactly the same positions with the same game results were the subject of analysis by both programs. In order to obtain as logical outcomes from particular positions as possible, and in order to particular heuristic values on average well reflect the expected game result, the games from World Chess Championship matches were particularly suitable, since:

- World Champions in general played strong chess moves in their World Championship matches, and committed relatively little mistakes,
- the contenders of the final match for the title of World Chess Champion were played by opponents of rather similar chess strength,
- large data set of games is available from these matches.

Of course, there were occasional turnovers in the course of the game in these matches, however, due to the large data set of positions available, statistical smoothing is expected to prevail. Besides, if the game ends in a win for the white player from a position that could be judged as completely lost for him (either by a human or by a highly negative heuristic value assigned by a computer), obviously some probability (albeit it is likely to be small) still exists that such result is possible between fallible players, in this case top chess players. Moreover, exactly the same positions with the same game results were the subject of analysis by both programs, which speaks for comparison between them being sensible.

The results in Fig. 8.15 were obtained with chess programs CRAFTY and RYBKA at search depth of 12 plies. They show the proportion of wins, draws, and losses achieved from positions where particular heuristic evaluation value was obtained by the program, for each program separately. The backed-up evaluation values were joined into intervals, so that each interval was represented by more positions. The chosen size of the interval was 0.10, the intervals ranging from -3.00 to 3.00 are shown for both programs.

The results show that backed-up evaluations of both programs tend to be monotonic with probability of winning in a game between fallible players. However, it is also obvious that the same backed-up evaluation values obtained by the evaluation functions of each program do not reflect the probability of winning in the same way. It suggests that the scales of heuristic evaluation values of the two programs are not the same. This is confirmed in Fig. 8.16, which shows the histograms where for each of the assigned intervals the number of positions that belong to particular interval is shown. Obviously, RYBKA's evaluation function tend to assign heuristic values that are more closely distributed around 0.

We view these results as another confirmation of our claim that the evaluation functions do *not* try to approximate some unknown "true" value of the position being evaluated (compare to Fig. 8.1). Each evaluation function may use its own range of heuristic evaluation values that enable the program to play in a direction-oriented way, while the backed-up evaluations that result from searching more deeply somehow reflect the probability of winning in a game of two fallible players of approximately equal strength.

8.4. Heuristic Evaluation Functions and Probability of Winning

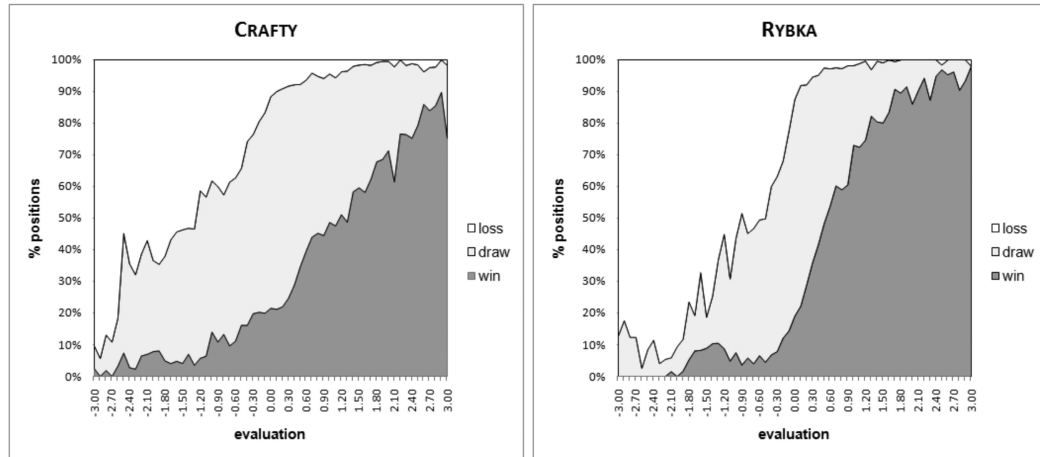


Figure 8.15: The proportion of wins, draws, and losses achieved for particular backed-up evaluation values at search depth of 12 plies with CRAFTY (left) and RYBKA (right).

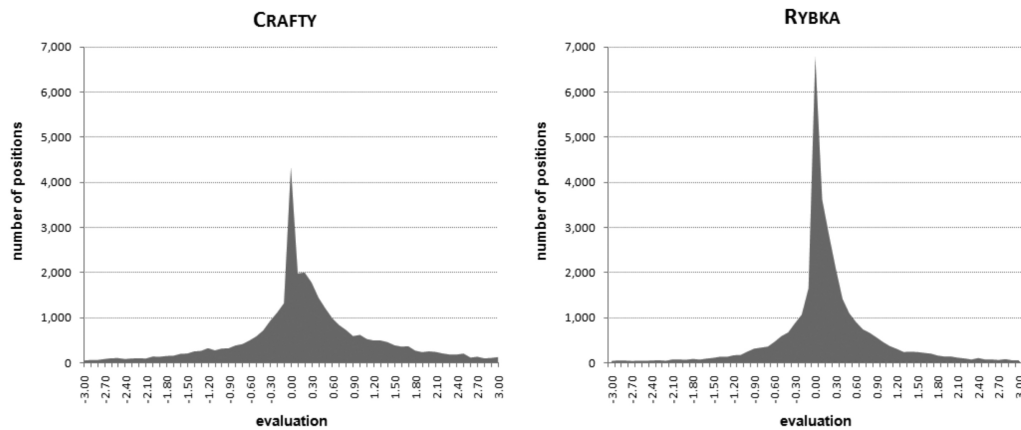


Figure 8.16: Number of positions that belong to particular backed-up evaluation values at search depth of 12 plies for CRAFTY (left) and RYBKA (right).

8.5 Possible Impacts on Theory and Practice

Three additional possible impacts of the monotonicity property of heuristic evaluation functions on the theory and practice of game playing are discussed. In Subsection 8.5.1, we argue that heuristic evaluations obtained by a search to different search depths are *not* directly comparable. In Subsection 8.5.2, we explain how the monotonicity property provides an explanation why in positions with a decisive advantage, best moves according to a chess program change less frequently with increasing search depth than they do in balanced positions. In Subsection 8.5.3, we show the usability of our findings for providing a heuristic-search based program with an ability to solve a difficult type of problems: detecting fortresses in chess.

8.5.1 Searching to Variable Depths Revisited

Having in mind the demonstrated monotonicity property of heuristic evaluation functions we could ask ourselves: “Are heuristic evaluations obtained by a search to different search depths really directly comparable?” Consider a minimax-based program searching to variable search depths. Due to various types of *search extensions* (extending interesting variations, *i.e.*, searching more deeply from seemingly more promising parts of the search tree), state-of-the-art chess programs frequently conduct a search to different depths of search. Afterwards, the backed-up evaluations are being compared in such way that the depth of search at which they were obtained is completely ignored.

However, in won positions, for example, backed-up heuristic values obtained from deeper searches should, on average, be expected to be higher due to the monotonicity property of heuristic evaluation functions. According to this observation, in such positions, if two moves result in approximately equal backed-up values, the one resulting from *shallower* search may well lead to a better decision. Obviously, the depth at which the backed-up evaluation was obtained is necessary to be taken into account in order to perform relevant comparisons of backed-up heuristic evaluation values (see Figure 8.17).

Let us illustrate this point by an example from a real game. Figure 8.18 shows CRAFTY’s backed-up evaluations obtained at search depths in range from 7 to 17 for two winning moves in the diagrammed position: 40.a5-a6 and 40.Nc7-e6. The evaluations tend to increase monotonically with increasing depth of search, indicating

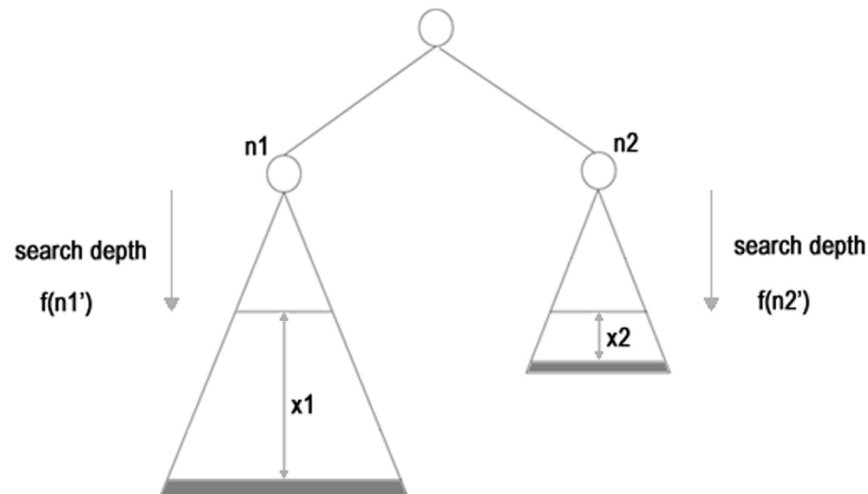


Figure 8.17: Let positions $n1$ and $n2$ be theoretically won (*i.e.*, won providing optimal play by both players). After searching to some arbitrary fixed depth d from the root position, backed-up heuristic evaluations $f(n1')$ and $f(n2')$ are obtained, so that $f(n1') < f(n2')$. Let the program continue the search for $x1$ more plies from the position $n1'$ and for $x2$ more plies from the position $n2'$, $x1 > x2$, leading to backed-up evaluations $f(n1'')$ and $f(n2'')$. Due to the monotonicity property of heuristic evaluation functions, $f(n1'') > f(n2'')$ could be expected, providing that the difference between $f(n1')$ and $f(n2')$ is sufficiently low, and the difference between $x1$ and $x2$ is sufficiently high. The program that completely ignores the depth of search where $f(n1'')$ and $f(n2'')$ were obtained will chose the move $n1$, although it obtained the lower evaluation after searching to the fixed depth of d . However, it is likely that in a game between two fallible players choosing the move $n1$ would result in lower probability of winning than if the move $n2$ was chosen.

that both moves lead to a win assuming optimal play. The program's move of choice at any depth is 40.a5-a6, that is, at each depth the evaluation of 40.a5-a6 is higher than the evaluation of 40.Nc7-e6. This indicates that in a game between two fallible players, the move 40.a5-a6 would lead to a higher probability of winning than the other move. Indeed, the move 40.Nc7-e6 would yield the black player some practical chances of escaping into a drawn king and two knights versus king endgame (it is also well known that exchanging pawns in won endgames, generally speaking, favors the weaker side). However, if the evaluation of the inferior move 40.Nc7-e6 was obtained at sufficiently higher depth of search than the evaluation of 40.a5-a6, CRAFTY would fail to make the best choice.

This observation is confirmed when currently the strongest chess engine according to the SSDF rating list [Kar08], RYBKA 3, is used for the analysis of this position.

8. MONOTONICITY PROPERTY OF HEURISTIC EVALUATION FUNCTIONS

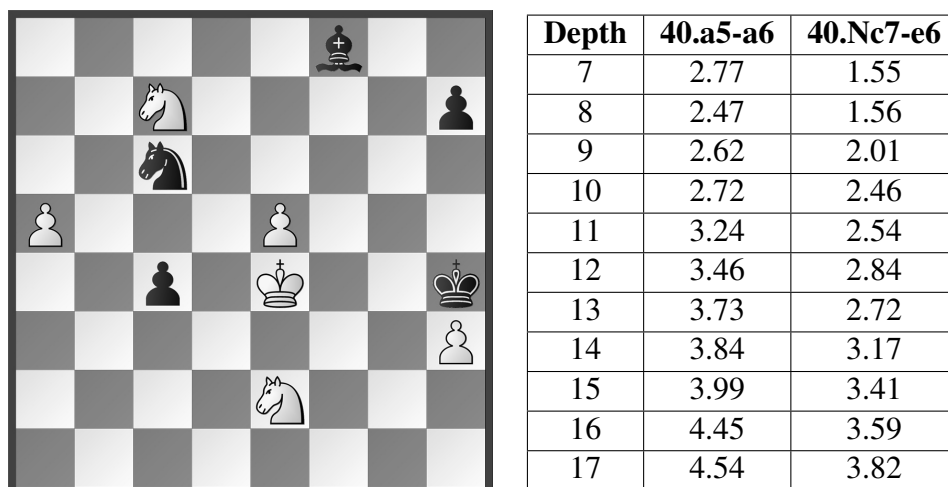


Figure 8.18: Botvinnik-Smyslov, World Chess Championship match (game 16, position after black's 39th move), Moscow 1954. White has two tempting continuations in this winning position: 40.a5-a6, keeping both white pawns on the board, and 40.Nc7-e6, attacking the black bishop. Both of them probably win, however, after 40.Nc7-e6 black can play 40...Nc6xa5!, and now if Black manages to sacrifice the knight for White's only remaining passed pawn, for example, after 41.Ne6xf8 (taking the bishop) 41...Na5-c6 42.Nf8xh7?? (taking the black pawn, but this is a mistake), Black saves himself with 42...Nc6xe5! 43.Ke4xe5 Kh4xh3, sacrificing the knight for achieving a drawn KNNKP endgame. In the game, Botvinnik played 40.a5-a6! and won five moves later. The table on the right shows CRAFTY's backed-up evaluations as results of search to different search depths.

As it is demonstrated in Figure 8.19, this program also finds 40.a5-a6 to be the best move, at any search depth. However, if the search depth used for evaluating the move 40.a5-a6 was less than 14 plies and the search for evaluating the move 40.Nc7-e6 was extended to 17 plies, RYBKA 3 would also choose the inferior move.

Searching to fixed depth (extended with quiescence search to ensure stable static evaluations) becomes particularly important when averaging the results of search over a large amount of data takes place. For example, when estimating human performance in problem solving. In Chapter 2, we conducted computer analyzes of World Chess Champions' games, aiming at a more objective assessment of chess playing strength of chess players of different times. The idea was to determine the chess players' quality of play (regardless of the game score), which was evaluated with the help of computer analyzes of individual moves made by each player. Basic World Champions' performance estimates were determined as the average differences between evaluations of moves that were played by the players and evaluations of best

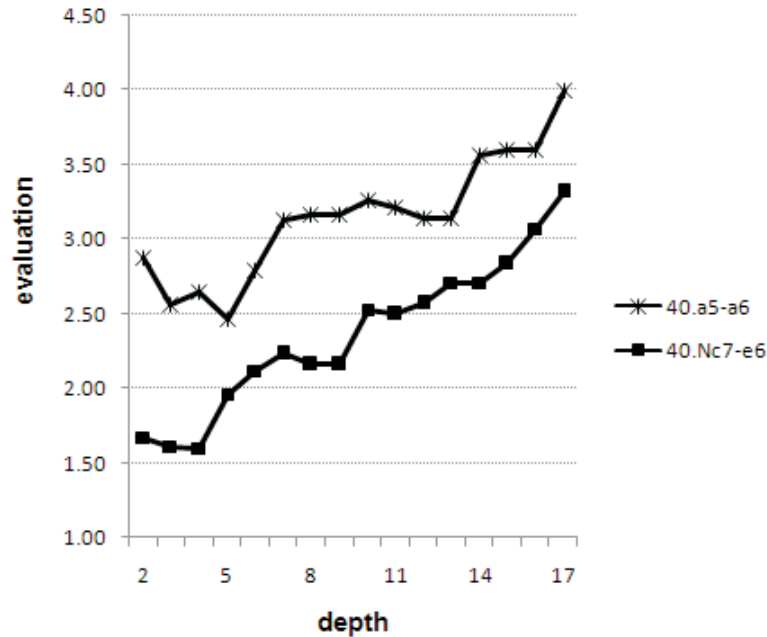


Figure 8.19: Backed-up evaluations with increasing depth of search from the position shown in Figure 8.18, obtained with currently the strongest chess engine, RYBKA 3, for the best two moves according to the program, *i.e.*, 40.a5-a6 and 40.Nc7-e6.

moves according to the computer. In this type of analysis, conducting search to variable depths (*e.g.*, by fixing the amount of time dedicated to search in an individual position) would likely lead to false conclusions, due to the monotonicity property of heuristic evaluation functions, as it is explained above.

Let us briefly return to blunder-rate measurements presented in Subsection 2.4.2. We labeled a move as a blunder when the numerical error as seen by CRAFTY at highest available depth exceeded the value of 1.00. According to the common interpretation of computer heuristic evaluations as understood by chess players (or chess programmers), the value of 1.00 in terms of differences between computer evaluations assigned to two particular chess positions, the latter resulting by making a move from the former one, is regarded as “losing a pawn without compensation” (with no other deficits of removing a pawn from the chessboard being taken into account) [GB06]. This interpretation seems very logical, since the common standard in programming evaluation functions in chess programs is that the value of a single pawn is 1.00.

However, taking into account the monotonicity property of heuristic evaluation

functions, this interpretation is false. The evaluations should necessary change with search depth, and so should the differences in evaluations between particular moves. We therefore cannot say that the value of 1.00 in terms of differences between evaluations represents “losing a pawn without compensation.”

The empirical results presented in this chapter strongly suggest the following answer to the question in the beginning of this subsection. Heuristic evaluations obtained by a search to different search depths are *not* directly comparable, in contrast to what is generally assumed both in literature and in practical applications.

8.5.2 Expectations of Decision Changes with Deeper Search

The monotonicity property of heuristic evaluation functions provides an explanation why in positions with a decisive advantage, best moves according to a chess program change less frequently with increasing search depth than they do in balanced positions (see Chapter 9). In the present chapter, we observed that in positions with a decisive advantage, backed-up evaluations of better moves according to the program on average increase more rapidly than backed-up evaluations of less good moves. This phenomenon can be most clearly seen in Figures 8.10 and 8.12. Since the backed-up evaluations of better moves on average increase more rapidly in positions with a decisive advantage, in such positions the differences between backed-up evaluations of candidates for the best move according to the program are likely to become bigger with increasing search depth, thus changes of programs’ decisions with increasing search depth are less likely to occur.

8.5.3 Detecting Fortresses in Chess

In this section, we will demonstrate a practical application of taking into account the monotonicity property of heuristic evaluation functions for solving a difficult type of problems that are currently regarded as unsolvable by using heuristic-search based programs: detecting fortresses in chess.^{††}

Fortresses are usually defined as positions when one side has a material advantage, however, the defender’s position is an impregnable fortress and the win cannot

^{††}We note that the detection of fortresses in chess is nowadays possible by using Monte-Carlo Tree Search (MCTS) [KS06; WBS08; Cou07]. However, the implementation of MCTS in current chess programs for the purpose of detecting fortresses only is likely to be rather impractical compared to the method that we propose in this subsection.

be achieved providing optimal play by both sides. Current state-of-the-art programs typically fail to recognize fortresses and seem to claim winning advantage in such positions, although they are not able to achieve actually the win against adequate defence. We will demonstrate that due to lack of monotonically increasing evaluations between successive depths that are otherwise expected in won positions, fortresses are detectable by using heuristic search to several successive search depths.

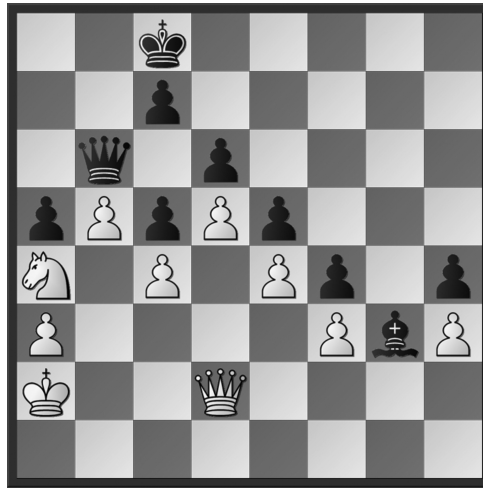


Figure 8.20: In this position the white player is to move, and has a winning positional advantage. State-of-the-art chess programs without an exception choose the move 1.Na4xb6 (white knight takes the black queen), which leads to big material advantage. However, after 1...c7xb6 (black pawn takes the white knight) black's position becomes an impregnable fortress and the win is no longer possible, providing adequate defence. Nevertheless, in the diagrammed position white has a winning plan: Ka2-b3, Na4-b2, Kb3-a4, and Qd2xa5.

The position in Fig. 8.20 is taken from the book *Dvoretsky's Endgame Manual* [Dvo08]. Current state-of-the-art chess programs without an exception chose to take the black queen with the knight (1.Na4xb6), which leads to big material advantage and to high evaluations that seemingly promise an easy win. However, it turns out that after 1...c7xb6 (black pawn takes the white knight) the backed-up evaluations cease to increase with increasing depth of search. In fact, black position becomes an impregnable fortress and the win is no longer possible, providing adequate defence. From the diagrammed position, grandmaster Dvoretsky gives the winning plan for the white player, which includes taking the insufficiently protected pawn on a5 (see the caption in Fig. 8.20). Had the programs chosen the proposed series of moves,

the backed-up evaluations would continue to increase monotonically with increasing search depth and the programs would find the way to win this position.

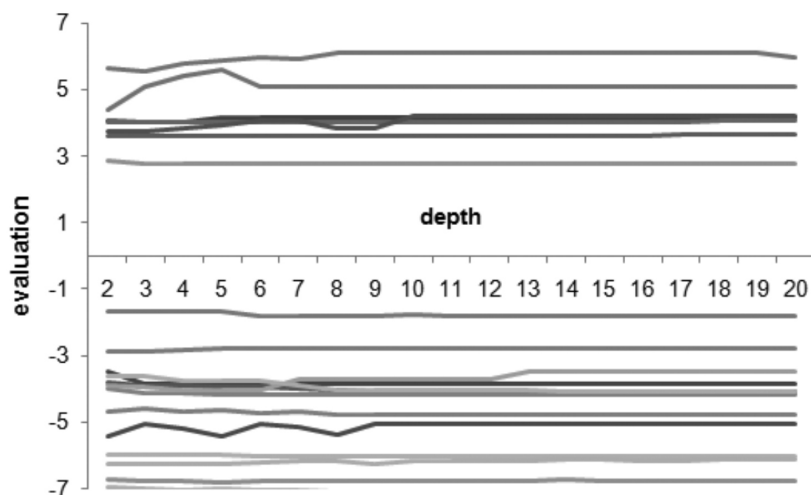


Figure 8.21: In the positions that could be regarded as fortresses, backed-up evaluations obtained by RYBKA cease to increase (or decrease) as it is otherwise expected in winning (losing) positions (compare to Figures 8.11 and 8.12).

We chose 20 positions from the aforementioned book by Dvoretsky that were recognized as fortresses for the following experiment. The positions were a subject of analysis by chess program RYBKA. The program's backed-up evaluations of searching to depths ranging from 2 up to 20 plies were obtained. Our claim was the following: Backed-up evaluations in positions that could be regarded as fortresses will not behave as it is usual for winning (losing) positions, that is they will not increase (or decrease) monotonically with increasing depth of search. The results of this experiment are demonstrated in Fig. 8.21 and they confirm this claim.

We should note that similar behavior of backed-up evaluation values as shown in Fig. 8.21 were obtained using various different chess programs for chess positions that are accepted as fortresses.

Chapter 9

Factors Affecting Diminishing Returns for Searching Deeper

This chapter is an updated and abridged version of the following publication:

1. Guid, M. and Bratko, I. Factors Affecting Diminishing Returns for Searching Deeper. *ICGA Journal*, Vol. 30, No. 2, pp. 75-84, 2007. [GB07]

In this chapter, we prove empirically that the rate of changed decisions that arise from search to different depths depends on (1) the quality of knowledge in evaluation functions, (2) the value of a node in the search space, and to some extent also on (3) the phase of the game.

9.1 Go-Deep Experiments and Diminishing Returns

Deep-search behavior and diminishing returns for additional search in chess have been burning issues for more than twenty five years in the game-playing scientific community. Two different approaches took place on this topic: *self-play* and *go-deep*. While in self-play experiments, two otherwise identical programs are matched with one having a handicap (usually in search depth), go-deep experiments deal with best-move changes resulting from different search depths of a set of positions.

Ken Thompson's pioneering work illustrated the benefits of increasing the search depth by conducting self-play experiments with chess program BELLE [Tho82]. In his experiments, the program searching to a fixed depth $d + 1$ scored a surprising 80% of the possible points against the program searching to a fixed depth of d plies

in a 20-game match. Some researchers extrapolated this work and speculated that a program searching to the depth of 12 plies would be able to achieve a rating that was higher than that of the World Champion [HACN90]. However, it seems highly unlikely that every additional ply of search leads to the same gain in playing strength. As Junghanns *et al.* point out, simple logic suggests that diminishing returns for searching deeper must eventually appear, illustrating this point with the following question: “Can there possibly be a big difference between a 100-ply program and one doing 101 ply?” [JSB⁺97]

The go-deep experiments were introduced for determining the expectation of a new best move being discovered by searching only one ply deeper. The approach is based on Newborn’s [New85] discovery that the results of self-play experiments are closely correlated with the rate at which the best move changes from one iteration to the next. Newborn [New85] formulated the following hypothesis. Let $RI(d + 1)$ denote the rating improvement when increasing the search depth from level d to level $d + 1$, and $BC(d)$ the expectation of finding a best move at level d different from the best move found at level $d - 1$, then:

$$RI(d + 1) = \frac{BC(d + 1)}{BC(d)} RI(d) \quad (9.1)$$

There were some objections about the above equation, *e.g.*, the one by Heinz [Hei98]: “Please imagine a chess program that simply switches back and forth between a few good moves all the time. Such behavior does surely not increase the playing strength of the program at any search depth.” He suggested that the discovery of “fresh ideas” looks like a much better and meaningful indicator of increases in playing strength than a best-move change at the next iteration of the search, and proposed “fresh best” moves instead, defined as new best moves which the program never deemed best before. Whatever the merit of this proposal, determining $BC(d)$ for higher values of d continued to be used in several experiments. In 1997, PHOENIX (Schaeffer, [Sch86]) and THE TURK (Junghanns *et al.*, [JSB⁺97]) were used to record best-move changes at iteration depths up to 9 plies. In the same year, Hyatt and Newborn [HN97] let CRAFTY search to an iteration depth of 14 plies. In 1998, Heinz [Hei98] repeated their go-deep experiment with DARKTHOUGHT. All these experiments were performed on somehow limited data sets of test positions and did not provide any conclusive empirical evidence that the best move changes taper off continuously with increasing search depth.

An interesting go-deep experiment was performed by Sadikov and Bratko [SB06]. They made very deep searches (unlimited for all practical purposes) possible by concentrating on chess endgames with a limited number of pieces. Their results confirmed that diminishing returns in chess exist, and showed that the amount of knowledge, which a program has, influences the precise time when diminishing returns will start to manifest themselves.

A remarkable follow-up on the previous work done on deep-search behavior using chess programs was published by Steenhuisen [Ste05] who used CRAFTY to repeat the go-deep experiment on positions taken from previous experiments to push the search horizon to 20 plies. He used the same experimental setup to search, among others, a set of 4,500 positions, from the opening phase, to a depth of 18 plies. His results show that the chance of new best moves being discovered decreases exponentially when searching to higher depths, and decreases faster for positions closer to the end of the game. He also reported that the speed with which the best-change rate decreases depends on the test set used.

The latter seems to be an important issue regarding the trustworthiness of the various results obtained by the go-deep experiments. How can one rely on statistical evidence from different go-deep experiments, if they obviously depend on the data set used?

In this chapter, we study experimentally additional factors which influence the behavior of diminishing returns that manifest themselves in go-deep experiments. Using a large data set of more than 40,000 positions taken from real games we conduct go-deep experiments with the programs CRAFTY, RYBKA, and SHREDDER to provide evidence that the chance of new best moves being discovered at higher depths depends on (1) the values of positions in the data set, (2) the quality of the evaluation function of the program used, and to some extent also on (3) the phase of the game.*

Among other findings, the results will demonstrate with a high level of statistical confidence that both “Best Change” and “Fresh Best” rates (as defined by Newborn [New85] and Heinz [Hei98], respectively) decrease with increasing search depth in each of the subsets of the large data set used in this study.

*In the paper *Factors Affecting Diminishing Returns for Searching Deeper* [GB07] we also showed that the chance of new best moves being discovered at higher depths depends to some extent also on the amount of material on the board. However, since the amount of material on the board is closely correlated with the phase of the game, we will omit those results in this thesis. An interested reader could find them in the aforementioned paper.

9.2 Experimental Design

The Chess programs CRAFTY, RYBKA, and SHREDDER[†] were used to analyze more than 40,000 positions from real games played in World Chess Championship matches. Each position occurring in these games after move 12 was searched to a fixed depth ranging from 2 to 12 plies. Similarly as in Section 8.2, searching was extended with *quiescence search* to ensure that programs' decisions were based on stable static evaluations.

For the measurements done in the go-deep experiments we use the same definitions as provided by Heinz [Hei98] and Steenhuisen [Ste05]. Let $B(d)$ denote the best move after a search to depth d , then the following best-move properties were defined.

Best Change $B(d) \neq B(d - 1)$

Fresh Best $B(d) \neq B(j) \forall j < d$

(d-2) Best $B(d) = B(d - 2)$ and $B(d) \neq B(d - 1)$

(d-3) Best $B(d) = B(d - 3)$ and $B(d) \neq B(d - 2)$ and $B(d) \neq B(d - 1)$

We give the estimated probabilities (in %) and their estimated standard errors SE (in Equation 9.2: $N(d)$ stands for the number of observations at search depth d) for each measurement of Best Change. The rates for Fresh Best, $(d - 2)$ Best, and $(d - 3)$ Best are given as conditional to the occurrence of a Best Change. We also provide mean evaluations of positions at each level of search.

$$SE = \sqrt{\left(\frac{BC(d)(1 - BC(d))}{N(d) - 1}\right)} \quad (9.2)$$

For confidence bounds on the values for best-change rates we use the 95%-level of confidence ($\lambda = 1.96$). We use the equation given by Steenhuisen [Ste05] (in Equation 9.3: m represents the number of successes in a sample size of n observations).

$$\frac{m + \frac{\lambda^2}{2} \pm \sqrt{m(1 - \frac{m}{n}) + \frac{\lambda^2}{4}}}{n + \lambda^2} \quad (9.3)$$

Similarly as in Section 8.2, we devised several groups of data in order to test our hypotheses. Properties of these groups are described at the beginning of each section.

[†]CRAFTY 19.2, and RYBKA 2.2n2 32-bit were used in the experiments. In Section 9.4, we also used DEEP SHREDDER 10 UCI and RYBKA 3 1-cpu 32 bit.

9.3 Diminishing Returns and Values of Positions

9.3.1 CRAFTY Goes Deep

Several researchers used CRAFTY for their go-deep experiments. However, none had such a large set of test positions at his disposal as we had (over 40,000 positions). Steenhuisen (2005) observed deep-search behavior of CRAFTY on different test sets and reported different Best Change rates and Best Change rate decreases for different test sets. This and the following section will show that best-change rates strongly depend on the values of the positions included in a test set.

As described in Section 8.2, to determine the best available approximation of the utility value of each analyzed position, the backed-up evaluation at the deepest search depth served as an “oracle”. We devised six different groups of positions based on their estimated utility values, as given in Table 9.1. In usual terms of chess players, the positions of Groups 1 and 6 could be labeled as positions with “decisive advantage”, positions of Groups 2 and 5 with “large advantage”, while Groups 3 and 4 consist of positions regarded as approximately equal or with a “small advantage” at most.

Table 9.1: Number of positions in each of the six groups of data. The groups were devised based on backed-up heuristic evaluation values obtained at search depth of 12 plies (CRAFTY).

Group	1	2	3	4	5	6
Evaluation (x)	$x < -2$	$-2 \leq x < -1$	$-1 \leq x < 0$	$0 \leq x < 1$	$1 \leq x < 2$	$x \geq 2$
CRAFTY	4,011	3,571	10,169	18,038	6,008	6,203

The results for each of the six groups are presented in Figure 9.1. The curves clearly show a different deep-search behavior of the program for the different groups, depending on the estimated value of positions they consist of. The chance of new best moves being discovered at higher depths is significantly higher for balanced positions than for positions with a decisive advantage. It is interesting to observe that this phenomenon does not yet occur at the shallowest search depths, while in the results of RYBKA it manifests itself at each level of search (see Section 9.3.2).

Tables 9.2 and 9.3 show the results for Groups 4 and 6. While the results resemble the ones obtained by Steenhuisen [Ste05] on the 4,500 positions in a sense that both

9. FACTORS AFFECTING DIMINISHING RETURNS FOR SEARCHING DEEPER

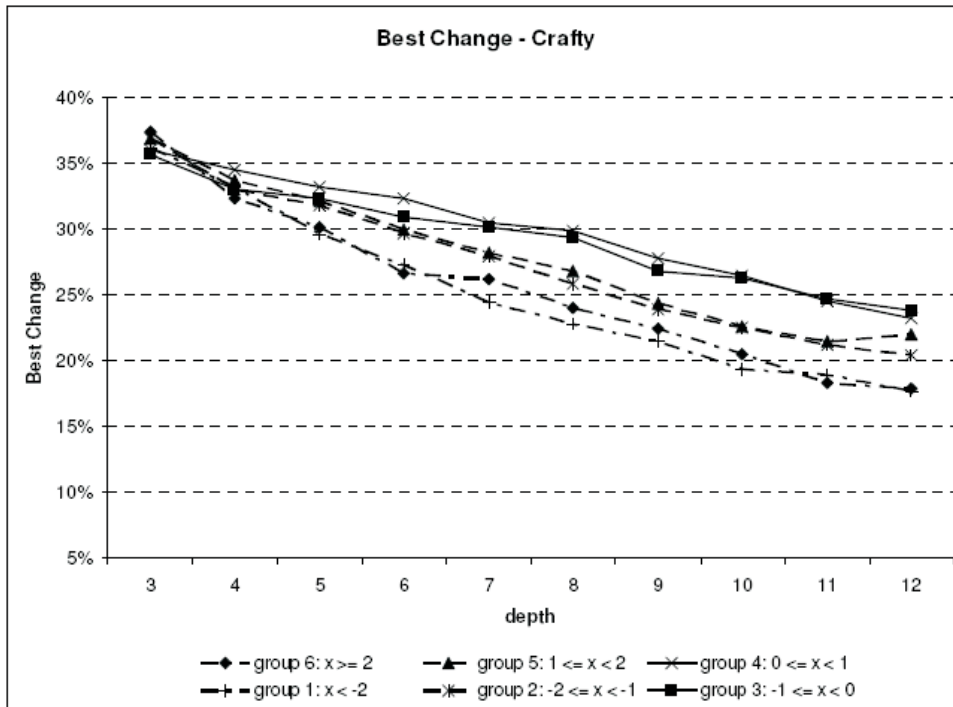


Figure 9.1: Go-deep results of CRAFTY on the six different groups of positions.

Table 9.2: Results of CRAFTY for the 18,038 positions of Group 4.

Search depth	Best Change in % (SE)	Fresh Best in %	(d-2) Best in %	(d-3) Best in %	Mean evaluation
3	35.96 (0.36)	100.00	-	-	0.36
4	34.47 (0.35)	74.88	25.12	-	0.37
5	33.18 (0.35)	64.16	27.34	8.50	0.37
6	32.34 (0.35)	54.38	28.44	11.38	0.37
7	30.48 (0.34)	49.53	31.14	9.51	0.37
8	29.86 (0.34)	42.81	31.45	11.27	0.38
9	27.75 (0.33)	40.02	33.87	10.81	0.38
10	26.48 (0.33)	37.77	33.31	10.57	0.38
11	24.53 (0.32)	34.79	33.48	11.14	0.38
12	23.17 (0.31)	32.26	33.07	12.04	0.39

Best Change and Fresh Best rates decrease consistently with increasing search depth, the rates nevertheless significantly differ for each of the two groups of positions.

The 95%-confidence bounds for Best Change (calculated using the Equation 9.2)

Table 9.3: Results of CRAFTY for the 6,203 positions of Group 6.

Search depth	Best Change in % (SE)	Fresh Best in %	(d-2) Best in %	(d-3) Best in %	Mean evaluation
3	37.42 (0.61)	100.00	-	-	2.64
4	32.27 (0.59)	73.93	26.07	-	2.76
5	30.13 (0.58)	64.85	24.83	10.33	2.84
6	26.60 (0.56)	55.70	28.06	9.70	2.95
7	26.21 (0.56)	49.88	27.37	10.52	3.04
8	23.99 (0.54)	39.92	31.18	11.02	3.17
9	22.44 (0.53)	37.21	32.18	12.72	3.29
10	20.47 (0.51)	36.30	30.79	11.50	3.42
11	18.30 (0.49)	31.37	32.42	12.07	3.54
12	17.85 (0.49)	29.27	29.99	13.91	3.68

at the highest level of search performed for the samples of 18,038 and 6,203 positions of Groups 4 and 6 are [22.56;23.97] and [16.91;18.82], respectively.

9.3.2 RYBKA Goes Deep

The results of go-deep experiments with RYBKA will not only confirm that Best Change rates depend on the values of the positions, but also demonstrate that the chance of new best moves being discovered at higher depths is lower at all depths compared to CRAFTY, which is rated more than 300 rating points lower on the Swedish (SSDF) Rating List [Kar08]. Table 4 presents the subsets evaluated by RYBKA, analogous to those presented in Table 9.1 and evaluated by CRAFTY.

Table 9.4: Number of positions in each of the six groups of data. The groups were devised based on backed-up heuristic evaluation values obtained at search depth of 12 plies (RYBKA).

Group	1	2	3	4	5	6
Evaluation (x)	$x < -2$	$-2 \leq x < -1$	$-1 \leq x < 0$	$0 \leq x < 1$	$1 \leq x < 2$	$x \geq 2$
RYBKA ₁₂	1,263	1,469	9,808	22,644	3,152	2,133

The results of RYBKA presented in Figure 9.2 resemble the results of CRAFTY in Figure 9.1, except that all the curves appear significantly lower on the vertical

9. FACTORS AFFECTING DIMINISHING RETURNS FOR SEARCHING DEEPER

scale. This result seems to be in line with the observation, based on the results by Sadikov and Bratko [SB06], that the amount of knowledge a program has (or the quality of the evaluation function) influences the deep-search behavior of a program. The big difference in strength of the two programs is likely to be the consequence of RYBKA having a stronger evaluation function; it is as well commonly known that chess players prefer evaluations of this program to RYBKA's evaluations. In their study, Sadikov and Bratko [SB06] claim that diminishing returns will start to manifest themselves earlier using a program with a stronger evaluation function, based on experiments performed on chess endgames, at the same time suspecting that similar results would be obtained with more pieces on the board. The results presented here seem to be in accordance with that claim.

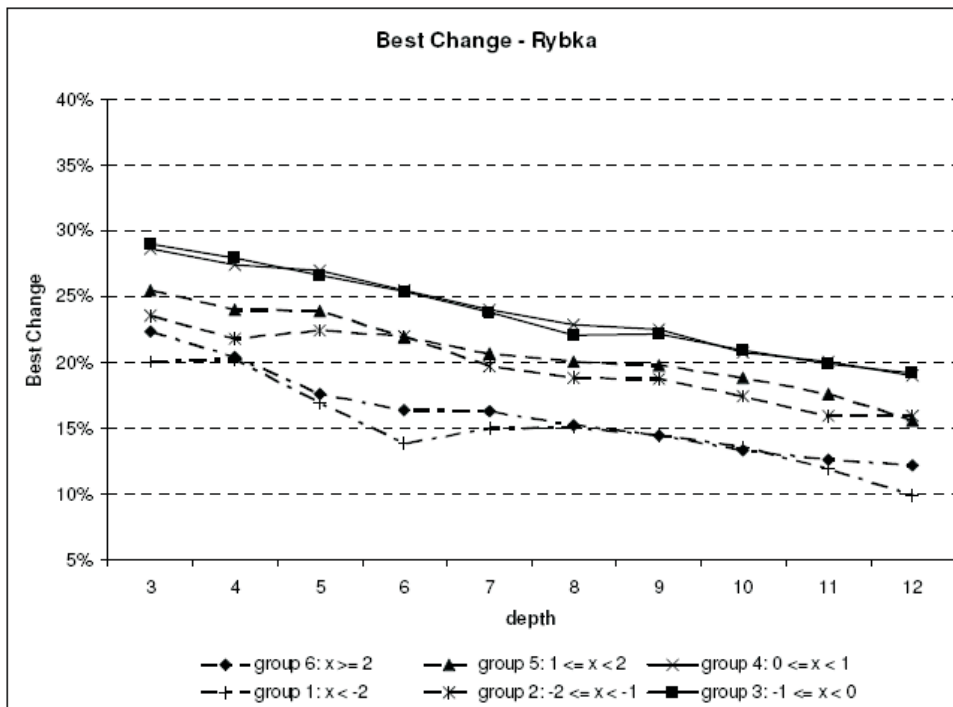


Figure 9.2: Go-deep results of RYBKA on the six different groups of positions.

Tables 9.5 and 9.6 (results of RYBKA) are the analogous of Tables 9.2 and 9.3 (results of CRAFTY). We will just briefly mention here that the mean evaluations of both programs in won positions monotonically increase with increasing search depth, which is in accordance with our findings presented in Chapter 8.

9.3. Diminishing Returns and Values of Positions

Table 9.5: Results of RYBKA for the 22,644 positions of Group 4.

Search depth	Best Change in % (SE)	Fresh Best in %	(d-2) Best in %	(d-3) Best in %	Mean evaluation
3	28.59 (0.30)	100.00	-	-	0.31
4	27.36 (0.30)	71.42	28.58	-	0.31
5	27.00 (0.30)	62.95	27.12	9.93	0.31
6	25.44 (0.29)	53.32	28.13	10.45	0.31
7	24.00 (0.28)	49.91	26.63	11.21	0.30
8	22.88 (0.28)	45.78	26.85	11.37	0.30
9	22.50 (0.28)	42.97	25.63	11.46	0.30
10	20.73 (0.27)	37.17	28.46	11.31	0.30
11	20.03 (0.27)	36.16	27.76	11.78	0.30
12	19.01 (0.26)	34.08	27.87	11.85	0.30

Table 9.6: Results of RYBKA for the 2,133 positions of Group 6.

Search depth	Best Change in % (SE)	Fresh Best in %	(d-2) Best in %	(d-3) Best in %	Mean evaluation
3	22.36 (0.90)	100.00	-	-	2.49
4	20.39 (0.87)	77.24	22.76	-	2.60
5	17.63 (0.83)	66.76	24.20	9.04	2.77
6	16.41 (0.80)	54.86	25.43	10.57	2.89
7	16.32 (0.80)	49.71	26.44	10.06	3.01
8	15.24 (0.78)	44.00	23.69	13.23	3.14
9	14.49 (0.76)	45.63	24.60	10.36	3.27
10	13.31 (0.74)	42.61	23.94	12.68	3.42
11	12.61 (0.72)	37.92	24.16	8.55	3.59
12	12.19 (0.71)	36.54	30.00	7.31	3.75

The 95%-confidence bounds for Best Change at the highest level of search performed for the samples of 22,644 and 2,133 positions of Groups 4 and 6 are [18.51;19.53] and [10.87;13.65], respectively.

9.4 Diminishing Returns and Quality of Evaluation Function

In this section, we give a comparison of Best Change rates of four programs of different strengths. In order to test our hypothesis that best-move changes correlate with the quality of the evaluation function of a program, we used another two programs, namely SHREDDER and the stronger version of RYBKA, RYBKA 3, to analyze more than 40,000 positions from the World Chess Championship matches, using the same methodology. We also did “SHREDDER goes deep” and “RYBKA 3 goes deep experiments”, and the results showed the same trends as those obtained by CRAFTY and RYBKA 2.

In order to provide a relevant comparison of chess strength of the four programs, relative to each other, we give in Table 9.7 the publicly available information from currently the latest (2009-04-10) version of the Swedish (SSDF) Rating List [Kar08], where ratings of several state-of-the-art chess programs are published. The ratings on this list are obtained after many games are played on the tournament level (40 moves in 2 hours followed by 20 moves in each following hour) between the programs, supervised by members of the Swedish Chess Computer Association (SSDF). ‘Games’ stands for the number of games on which the rating is based, and ‘Against’ for the average rating of opponents. The ‘+’ and ‘-’ denote respectively the upper and lower 95% confidence intervals. Although the versions of the programs slightly differ from the ones that we used in our experiments, the listed ratings should give enough information about their strength, in particular relative to each other.

Table 9.7: Comparison of the four programs that we used in our experiments according to the SSDF rating list (slightly different versions were used in our experiments). All four programs listed here were ran on 256MB Athlon 1200 MHz.

Program	Rating	+	-	Games	Win %	Against
DEEP RYBKA 3	3073	44	-44	253	55	3039
RYBKA 2.3.1 Arena	2923	23	-23	920	53	2904
SHREDDER 10 UCI	2827	20	-20	1246	58	2769
CRAFTY 19.17	2527	41	-44	304	30	2677

In Section 8.4 we showed that the scales of heuristic evaluation values of different programs may not be the same, that is, each evaluation function may use its own

range of heuristic evaluation values. Having this observations in mind, in order to perform a relevant comparison of programs' Best Change rates, we now observed Best Change rates on all available positions. Moreover, the data sets of positions analyzed were exactly the same. The results are given in Fig. 9.3. The curves of Best Change rates are ordered in accordance with the ratings of the programs. RYBKA 3, which is regarded as the program with the strongest evaluation function of the three programs, has the lowest Best Change curves, and the opposite applies to CRAFTY, of which the evaluation function is considered to be the weakest one. Qualitatively, quite similar results were observed for minor subsets of data as well.

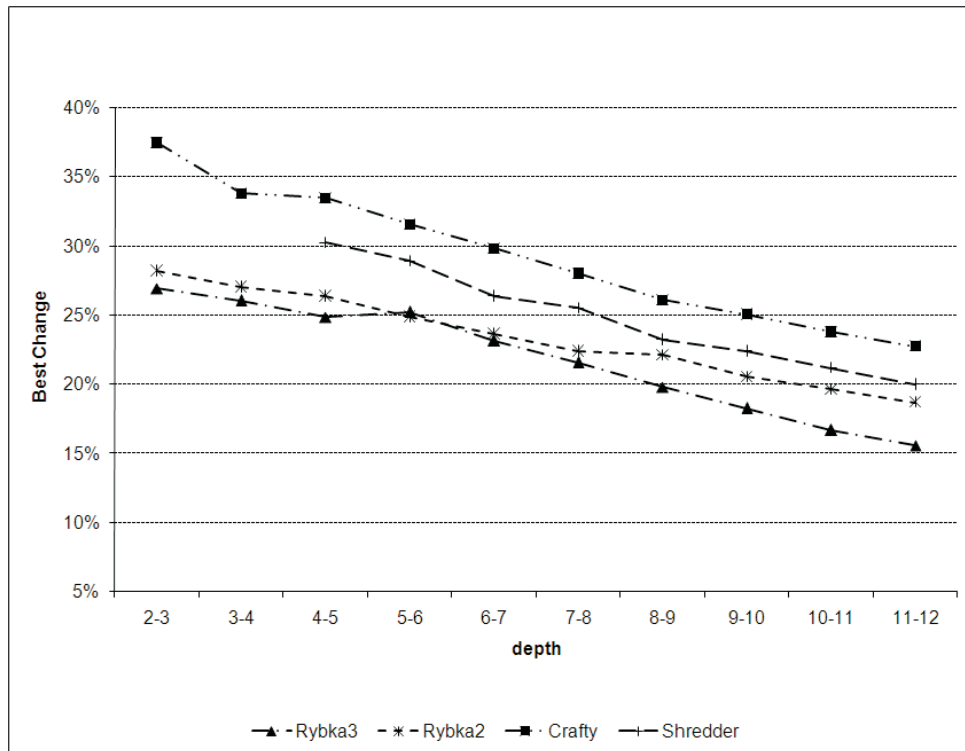


Figure 9.3: Go-deep results on all available positions with CRAFTY, SHREDDER, RYBKA 2, and RYBKA 3. Positions that were not analyzed by all four programs were excluded.

Could Best Change rates be used as a direct measure of the quality of an evaluation function? This question requires a further investigation. Consider a program that always selects the first move of the alphabetically sorted possible moves. In such a case, the Best Change curve coincides with the horizontal axis (which also happens in case of a “perfect” evaluation), despite of terribly low quality of such program’s

evaluations. However, this is a rather contrived case. As our experimental results suggest, evaluation functions of successful programs may have some properties that make comparison based on Best Change rates sensible. In each case, we suggest that a data set used for such an investigation should be representative for the whole game of chess, as it was probably the case with our large data set of real-game positions.

9.5 Diminishing Returns and Phase of the Game

Steenhuisen [Ste05] was the first to point out that the chance of new best moves being discovered at higher depth decreases faster for positions closer to the end of the game. However, having in mind that deep-search behavior depends on the values of positions in a test set, it seems worthwhile to check whether his results were just the consequence of dealing with positions with a decisive advantage (at least on average) in a later phase of the game. For the purpose of this experiment we took only a subset with more or less balanced positions with depth 12 and an evaluation in the range between -0.50 and 0.50 (see Table 9.8). Our results show that in the positions that occurred in the games later than move 50, the chance of new best moves being discovered indeed decreases faster, which agrees with Steenhuisen's [Ste05] observations. The experiments in this and the following section were performed by CRAFTY.

Table 9.8: Five subsets of positions of different phases in the game, with evaluations obtained at search depth 12 in range between -0.50 and 0.50 (CRAFTY).

Group	1	2	3	4	5
Move no. (m)	$m < 20$	$20 \leq m < 30$	$30 \leq m < 40$	$40 \leq m < 50$	$m \geq 50$
Positions	7,580	6,106	3,418	1,356	961

The results presented in Figure 9.4 show that while there is no obvious correlation between move number and the chance of new best moves being discovered at higher depth, in the positions of Group 5 that occurred closer to the end of the game the Best Change curve nevertheless appears lower than the curves of the other groups. Table 9.9 shows the best-move properties for this group.

9.5. Diminishing Returns and Phase of the Game

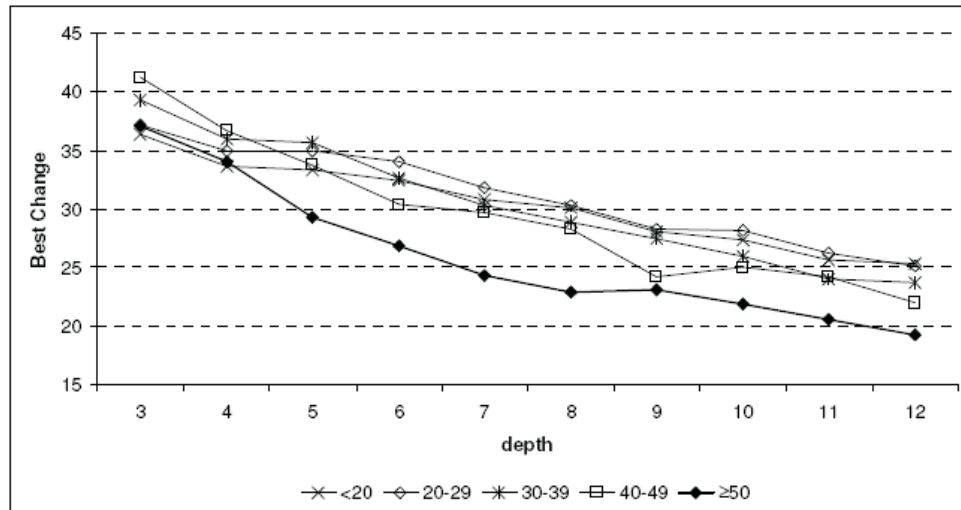


Figure 9.4: Go-deep results with positions of different phases of the game.

Table 9.9: Results for the 961 positions of Group 5.

Search depth	Best Change in % (SE)	Fresh Best in %	(d-2) Best in %	(d-3) Best in %	Mean evaluation
3	37.04 (1.56)	100.00	-	-	0.07
4	34.03 (1.53)	72.78	27.22	-	0.05
5	29.24 (1.47)	60.85	27.40	11.74	0.05
6	26.85 (1.43)	49.22	30.23	14.34	0.03
7	24.35 (1.39)	47.44	29.91	10.26	0.02
8	22.89 (1.36)	45.91	27.27	9.55	0.02
9	23.10 (1.36)	38.29	32.88	10.81	0.02
10	21.85 (1.33)	37.62	27.62	11.43	0.02
11	20.60 (1.31)	33.33	32.83	12.12	0.02
12	19.25 (1.27)	26.49	36.22	8.65	0.01

The 95%-confidence bounds for Best Change at the highest level of search performed for the sample of 961 positions of Group 5 are [16.88;21.86].

Chapter 10

Conclusions

10.1 Critical Analysis and Open Questions

We view this thesis as a progress report on work that needs to be advanced in a great deal of directions. We do believe it describes some novel aspects on the comparison and combination of search and knowledge in human and machine problem solving, and that it provides one possible view on possibilities of developing heuristic-search methods for evaluating and improving problem-solving performance. Our hope is that the research done in the scope of this thesis will stimulate further extensions, emendations, or even refutations.

We are well aware that we have limited ourselves to only a small portion of references that exist in such a large scientific area such as Human and Machine Problem Solving. Only a rather small part of highly related branches in psychology such as the huge field of *Integrated Cognitive Architectures*, and a rather small part of an important and highly related field of *Artificial Intelligence in Education*, particularly with respect to building intelligent tutoring systems (an excellent book on the subject was recently written by B.P. Woolf [Woo08]) were addressed. We do hope, however, that researchers of all kind of areas related to Human and Machine Problem Solving will find in our work at least some fresh ideas for arriving farther on our common way in the enormous “problem space” that we are all searching through.

The research done in the scope of the thesis was performed in the framework of human and computer game playing, and the game of chess was used as the experimental domain. While we advocated that many researchers used the game-playing platform and the domain of chess in their experiments, the explicit or implicit mes-

10. CONCLUSIONS

sage of their works being that the results for chess are generalizable to other domains, we actually did not provide any specific evidence that our work is extendable outside the scope of the game of chess. This is clearly one possible future research direction.

There is almost a countless number of (even fundamental) scientific questions that remain open. Below we mention five of them.

1. How to define a measure of understandability (or comprehensibility) of a body of knowledge (a theory, model, problem-solving strategy) for a human? This is an old fundamental question of AI that has been rarely considered, and for which not even a reasonable tentative solution exist.
2. What are the characteristics of human-understandable (or human-assimilable) representations of a theory, model, or problem-solving strategy? There are limits on memory – what can be memorized, on search complexity: How many inference steps or moves a human is able to look ahead? There should be a well balanced mixture of knowledge requirements, useful key patterns, and some search (not too much knowledge, not too much search).
3. What is an effective “conceptualization” of a domain theory for a human? One that can be effectively learned, memorized, and effectively applied for problem solving?
4. How to generate human-meaningful and natural commentary, depending on a student’s level and type of knowledge and skill? A human may be solving the task using a particular strategy, so the commentator should be aware of this and make comments relative to this strategy; or alternatively, the commentator may be aware of a student’s limited knowledge (limited problem-solving strategy, limited declarative knowledge) and may decide to draw a student’s attention to another strategy that is better applicable to the problem at hand.
5. How to define a measure of cognitive difficulty of a given problem? Is it measurable at all or is it unique to each individual?

We address some more open questions and suggestions for further research directions in the sequel where assessments of the contributions of the thesis are stated.

10.2 Search and Knowledge for Estimating Human Problem Solving Performance

In Part I of the thesis, “Search and Knowledge for Estimating Human Problem Solving Performance,” we addressed the following question: “how can we develop methods based on computer heuristic search for evaluating human problem solving performance?” Chapters 2 and 3 answer the research question 1 (RQ1): “how can a computer be used to assess a human’s problem-solving performance?” Chapter 4 answers the research question 2 (RQ2): “how can a machine problem solving model be used to assess the difficulty of a set of given problems for a human?”

In Chapters 2 and 3, our aim was to demonstrate that heuristic-search based programs can be useful estimators of human problem-solving performance. We introduced a novel method based on computer heuristic search, for evaluating such performance at solving a given set of problems, and provided an analysis of appropriateness of this method. As a case study of estimating problem-solving performance using a heuristic-search based program we devised a comparison of World Chess Champions, top-level human experts in the game of chess. It is based on the evaluation of the games played by the World Chess Champions in their championship matches. We were interested in the chess players’ quality of play regardless of the game score, performing computer analyzes of individual moves made by each player. As the basis for estimating the chess players’ problem-solving performance we chose the average differences between heuristic evaluations of their decisions and heuristic evaluations of computer’s decisions, all of them obtained on a large set chess positions and at some fixed depth of search.

In Chapter 3, we confirmed the trustworthiness of our approach to estimating problem-solving performance in chess. We demonstrated that at least for pairs of the players whose scores differ significantly, it is not very likely that their relative rankings would change if (1) a stronger chess program was used, or (2) if the program would search more deeply, or (3) larger sets of positions were available for the analysis. The results show that, at least for the players whose scores differ significantly, the rankings are surprisingly stable over a large interval of search depths, and over a large variation of position sample. Even extremely shallow search of just two or three ply enable reasonable mutual rankings for some pairs of the players. Different chess programs were used to provide further evidence that using other, stronger

chess programs would be likely to result in similar rankings of the players whose scores differ by more than an average margin. Experimental results and theoretical explanations were provided to show that, in order to obtain a sensible ranking of the champions' performances by applying the proposed method it is not even necessary to use a computer program that is stronger in terms of chess strength than the champions themselves.

Generally speaking, the results of our computer analysis can be nicely interpreted by a chess expert. Some of the results may be considered also as an interesting contribution to the field of chess. Capablanca's outstanding result in terms of mean value loss will probably appear to many as such an interesting finding, although it probably should not come as a complete surprise.

Our goal was to devise a method for estimating problem-solving performance that would lead to the correct ranking of problem solvers' performances, and our assumption was that the computer estimator of problem-solving performance is trustworthy already when it is equally unfair to all problem solvers. The latter issue was not addressed in our work, and the following question remains unanswered: Does the program's style of play exhibit preference for the styles of any particular players?

Various discussions about our ICGA Journal publication [GB06] took place at different places, including scientific ([Haw07]) as well as popular blogs and forums across the internet. A shortened version of that publication was published by the popular chess website, Chessbase.com (see [Che06b]). The same website soon published some interesting responses by various readers from all over the world, including some by scientists (Chessbase.com, see [Che06a]). One frequent question by the readers was associated with the meaning of the players' scores according to our basic criterion. A typical misinterpretation of their meaning went like this: "For every 8 moves on average, CRAFTY expects to gain an advantage of one extra pawn over Kasparov" (Chessbase.com, [Che06a]). As we mentioned in Chapter 3, the scores obtained by the program only measure the average differences between the players' choices of move and the computer's choice. The experimental results presented in this thesis demonstrate that the scores are associated with the strength of the program and are not invariable neither for the same program at different depths of search, nor for different programs at the same depth of search. However, as the analysis shows these scores that are relative to the computer used, have good chances to produce sensible rankings of the players.

There were also speculations that using chess programs stronger than CRAFTY

for the analysis would lead to completely different results, and that Capablanca's result is merely a consequence of his style being similar to CRAFTY's style. In the thesis, we used three chess programs stronger than CRAFTY as estimators of problem-solving performances of the World Chess Champions and confirmed our previous findings. We observed that the three champions whose scores differ sufficiently from the others remained on their positions at any level of search using any of the program, when comparing their scores to the average scores of the rest of the champions.

There is much space for possible improvements of criteria for estimating chess-players' performances. For example, instead of measuring absolute differences between heuristic evaluations of chess-players' decisions and heuristic evaluations of computer's decisions, it would perhaps be more accurate to express these differences in terms of deteriorated probability of winning the game, as we suspect that computer-chess programs' heuristic evaluations are not on interval scale. That is, the difference of, *e.g.*, 0.60 at some particular search depth is much less meaningful in won/lost positions than in positions of (approximate) equality. Such an approach would also make the arbitrary set boundaries (-2.00 and 2.00) unnecessary. Second and different approach to estimating human problem-solving performance could be found in the works by Guy Haworth (see for example [Haw07] and [HRdF10]).

In Chapter 4, a heuristic-search based method was designed for assessing the average difficulty of a given set of chess positions. This made it possible to compare players of different playing styles. We demonstrated that the outstanding score of José Raúl Capablanca, the 3rd World Champion, in terms of the average deviation between evaluations of played moves and best-evaluated moves (the basic criterion) should be interpreted in the light of the comparatively low difficulty of positions in Capablanca's games. This is quite in line with the known assessments in the chess literature of his style.

Our measure of position difficulty seems to have produced sensible results. These results are qualitatively much in line to the observation of how an expert chess commentator would describe the players in this study in terms of their playing style. We believe that no published related work regarding using heuristic-search estimator of difficulty of chess positions exist. As a line of future work, it would be interesting to explore by means of a psychological study, how well our difficulty measure reflects the true cognitive difficulty of a given set of chess positions. Related open question is: How to take into account the differences between players in the average difficulty of the positions encountered in their games? We also believe that the algorithm for

measuring the difficulty could be further improved, for example, by assigning bigger weights to changes in decisions at greater search depths. Finally, there is an open question how to adjust this algorithm to be suitable for accessing the difficulty of *tactical positions* where quiescence search often does most of the job, leading the computer to find the best solution of an otherwise difficult problem from a human's point of view in merely a tiny portion of a second of time (and possibly even at the shallowest depths, making our difficulty measure irrelevant).

10.3 Search and Knowledge for Improving Human Problem Solving Performance

In Part II of the thesis, “Search and Knowledge for Improving Human Problem Solving Performance,” we addressed the following question: “how can we develop methods based on computer heuristic search for improving human problem solving performance?” Chapters 5 and 6 answer the research question 3 (RQ3): “how can machine problem solving be used in tutoring, for teaching a human to solve problems in a given problem domain?” Chapter 7 answers the research question 4 (RQ4): “how can knowledge represented in a form suitable for the computer, be transformed into a form that can be understood and used by a human?”

In Chapter 5, we presented a novel approach for automated generation of human understandable comments on decisions in problem solving. Our approach is based on a computer program that uses heuristic search. We set a quite ambitious target domain: the complete domain of chess. The long-term goal of our research is to develop a computer system that will provide commentary of chess moves and possible continuations in a comprehensible, user-friendly and instructive way, thus using the power demonstrated by the ever stronger chess programs for the purposes of annotating. We demonstrated the main advantages of our approach over the related proposals, namely:

- the ability to annotate chess games during all the phases of the game,
- the automatically generated commentary, aside from the ability to comment on tactical positions, also expresses a solid understanding of strategic concepts behind variations that chess programs suggest in given positions.

There is an array of cognitive issues still to be solved to arrive at really good annotating system, *e.g.*, when to comment and when not, which move (the best or some similar but better) to compare the move to, and what is the most suitable depth of lookahead, to name but a few. These are obvious tasks for future work.

In order to make possible for annotating software to make more in-depth comments, obtaining knowledge for construction of more complex positional features is desirable. In Chapter 6 we demonstrated our novel approach, based on argument-based machine learning [MvB07], to the formalization of complex patterns for the purpose of commenting on problem solving decisions and/or intelligent tutoring. We investigated a particular aspect in the development of a chess annotating software - the ability of making intelligent comments on the positional aspects of a chess game. This task is made more difficult by the fact that the strength of the chess playing programs mainly comes from search and not from subtle positional knowledge which is necessary for generating positional comments. Therefore, components of a chess program's evaluation function are not sufficient for making in-depth positional comments. Defining deep positional patterns requires powerful knowledge-elicitation methods. Our study suggests that argument-based machine learning enables such a method.

Our future work will be associated with further improvements of our annotation software. We intend to implement several additional positional features into its evaluation function, in order to make the commentary more instructive. In particular, the expert module of our annotation system, which provides the user with a commentary of chess games, based on learned or manually-crafted positional features, and possibly with more detailed explanations about particular features of chess positions, requires further attention. As part of future work, we intend to apply this knowledge-acquisition method to the formalisation of other positional concepts of fuzzy nature, such as weak or strong pawn structures, pressure on the opponent's king, space advantage, harmony among the pieces, etc.

In Chapter 7, we demonstrated a procedure for semi-automatic synthesis of knowledge usable for intelligent tutoring purposes. We developed an approach to deriving meaningful concepts and strategies usable for constructing a heuristic evaluation function ready to be used for commenting on problem-solving decisions. As a case study, we semi-automatically synthesized textbook instructions for teaching the difficult king, bishop, and knight versus the lone king (KBNK) endgame. Moreover, we used the obtained goal-based rules as a heuristic evaluation function to produce ex-

ample games containing automatically generated instructions. The derived strategy was found to be suitable for educational purposes at the level targeted for. Among the reasons to support this assessment was that the instructions “clearly demonstrate the intermediate subgoals of delivering checkmate.”

Our procedure for semi-automatic synthesis of knowledge combines ideas from argument-based machine learning with specialized minimax search to extract a strategy for solving problems that require search. Using this approach, a domain expert and a machine learning algorithm improve the model iteratively. It is particularly suitable from the expert’s point of view that argument-based machine learning offers several advantages for knowledge elicitation:

- it makes easier for the experts to articulate their knowledge,
- it facilitates the experts to adjust the level of introduced concepts to be acquired by students as dictated by the level of skills of students targeted at,
- the experts only need to provide relevant knowledge, and
- the obtained knowledge is
 - consistent with expert knowledge,
 - in a form suitable for use in a computer tutoring application,
 - in a form that can be understood and used by a human.

We also explained the guidelines for the interaction between the machine and the expert in order to obtain a human-understandable rule-based goal-oriented model for teaching how to solve problems in a particular symbolic domain, and how the instructions, including illustrative diagrams, could be derived semi-automatically from such a model.

Deriving human-understandable concepts and strategies from chess tablebases is an actual AI challenge. The positive outcome on the human understandability of the derived concepts and strategies would represent a milestone. However, the value of the demonstrated approach yet needs to be proven. So far we only have the positive opinion of chess coaches who commented on the derived strategy. There are at least two obvious directions for future work. First, to create a computer tool for teaching the KBNK endgame. Second, to use the described procedures to synthesize instructions for another, preferably much harder endgame.

10.4 On the Nature of Heuristic Search in Computer Game Playing

In Part III of the thesis, “On The Nature of Heuristic Search in Computer Game Playing,” we aimed at improving the understanding of properties of heuristic search and consequences of the interaction between search and knowledge that typically occurs in both human and machine problem solving.

In Chapter 8, we analyzed the properties of successful evaluation functions in game playing. *Monotonicity property* of heuristic evaluation functions for games was revisited. This is a property of successful heuristic evaluation functions for games. Namely, that backed-up values of the nodes in the search space have to tend to approach monotonically to the terminal values of the problem state space with the depth of search. The experimental results show that the evaluation functions used in typical chess programs tend to have this property that enables the program to play with a sense of direction towards a desirable goal. We demonstrated that backed-up heuristic values therefore do not approximate some unknown “true” or “ideal” heuristic values with increasing depth of search, in contrast to what is generally assumed in the literature, and point out that heuristic evaluation functions should *not* respect the minimax relation. That is, backed-up heuristic evaluation values should not be invariant along the game tree, as game-theoretical values in the theoretical minimax model are.

We demonstrated a practical application of taking into account this desirable property of heuristic evaluation functions for solving a difficult type of problems: detecting fortresses in chess. The experimental results also show that backed-up heuristic evaluation values in chess programs tend to be monotonic in the probability of winning a game between two fallible players. We discussed some of possible impacts of the monotonicity property of heuristic evaluation functions on the theory of game playing, and pointed out that heuristic evaluations obtained by searching to different search depths are not directly comparable, in contrast to what is generally assumed both in literature and in practical applications.

In Chapter 9, we studied experimentally additional factors which influence the behavior of diminishing returns that manifest themselves in the so-called *go-deep* experiments. Deep-search behavior and the phenomenon of diminishing returns for additional search effort have been studied by several researchers, whereby different

10. CONCLUSIONS

results were obtained on the different data sets used. Our results were obtained on a large set of more than 40,000 positions from real chess games using programs CRAFTY, RYBKA, and SHREDDER. They provide an empirical proof that the rate of changed decisions that arise from search to different depths depends on:

1. the quality of knowledge in evaluation functions, and
2. the true value (relative to a fixed search depth) of a node in the search space.

Among other findings, the results also demonstrated with a high level of statistical confidence that changes of the programs' decisions decrease with increasing search depth in each of the subsets of the large data set used in our experiments.

Could the rates of changed decisions that arise from search to different depths be used as a direct measure of the quality of an evaluation function? If so, under what conditions? These questions require a further investigation and in our opinion represent an interesting line of future work.

Dodatek A

Razširjeni povzetek v slovenskem jeziku (Extended Abstract in Slovene Language)

Znanje in preiskovanje pri človeškem in računalniškem reševanju problemov

A.1 Uvod

V umetni inteligenci obstajajo formalizirani in algoritmizirani pristopi k reševanju problemov, ki so po svoji naravi kombinatorični, kot je npr. načrtovanje operacij ali igranje miselnih iger. V teh pristopih so problemi predstavljeni s problemskim prostorom, ki je značilno neke vrste graf, reševanju problema pa ustreza preiskovanje ustreznega grafa. Zaradi kombinatorične zahtevnosti se ti problemi rešujejo s hevrističnimi metodami preiskovanja, pri katerih problemu specifične hevristike predstavljajo znanje o konkretni problemski domeni. Tako ti računalniški pristopi v umetni inteligenci v grobem vsebujejo dve komponenti: specifično *znanje* o problemu ter *preiskovanje* med alternativami.

Metode hevrističnega reševanja problemov so tudi dober model človeškega reševanja problemov, ki prav tako odraža ti dve komponenti: preiskovanje in znanje o konkretnem problemu. Seveda pa sta pri računalnikih in ljudeh ti dve komponenti zelo različno zastopani. Človek – ekspert tipično uporablja veliko bogatejše znanje o samem problemu, medtem ko je prednost računalnika v neprimerno hitrejšem preiskovanju. Pionirsko delo o modeliranju človeškega reševanja problemov z računalniškim sta opravila v svoji dobro znani raziskavi A. Newell in H.A. Simon [NS72].

Doktorske disertacija obravnava možnosti razvoja metod, temelječih na hevrističnem preiskovanju, za ocenjevanje in izboljševanje uspešnosti pri človeškem in računalniškem reševanju problemov. Vsebuje tudi prispevke k splošnemu razumevanju lastnosti hevrističnega preiskovanja ter posledic interakcije med znanjem in preiskovanjem, tipično prisotne pri reševanju problemov, tako pri ljudeh kot pri računalnikih.

V tem uvodnem delu bomo najprej na kratko predstavili reševanje problemov, hevristično preiskovanje in interakcijo med preiskovanjem in znanjem. Nato bo opisana primernost okolja igranja iger kot platforme za raziskovalno delo pri umetni inteligenci in šah kot ustrezna raziskovalna domena. Na koncu uvoda predstavljamo osrednja znanstvena vprašanja, povezana z našim raziskovalnim delom in prispevke k znanosti.

A.1.1 Reševanje problemov in hevristično preiskovanje

Človek je soočen s problemom, če želi nekaj in ne ve takoj, katero sosledje dejanj lahko izvede, da to doseže [NS72]. Pri umetni inteligenci je tipična splošna shema za predstavitev problemov imenovana prostor stanj. Prostor stanj je graf, katerega vozlišča ustrezajo problemskim situacijam, dani problem pa je omejen na iskanje ustrezne poti v tem grafu.

Preiskovanje grafov za namene reševanja problemov vodi do problema kombinatorične kompleksnosti, ki izhaja iz hitro naraščajočega števila pojavljajočih se alternativ. Za premostitev tega problema se navadno uporablja hevristično preiskovanje. Vozliščem v grafu se določi hevristične ocene, ki odražajo obetavnost vozlišč za doseganje ciljnih stanj. Namen je izvajati preiskovanje iz najbolj obetavnih vozlišč med možnimi kandidati [Bra00]. Na splošno hevristike predstavljajo strategije, ki s pomočjo uporabe informacij kontrolirajo reševanje problemov, s katerimi se soočajo tako ljudje kot računalniški sistemi [Pea84].

A.1.2 Preiskovanje in znanje

Hevristično preiskovanje implicira tako uporabo preiskovanja kot znanja. Znanje je moč uporabiti pri usmerjanju preiskovanja in preiskovanje je lahko v pomoč pri osvajanju znanja. Hitrejši program lahko preišče več vozlišč in torej išče globlje. To sugerira možno rešitev pri reševanju številnih tipov problemov: pridobiti hitrejšo strojno opremo. Drug pristop k reševanju problemov je zanašanje na kakovostno znanje za natančno ocenjevanje problemskih stanj z uporabo manj preiskovanja. V ekstremnem primeru računalniški program (ali človek) z dovolj globokim razumevanjem problemske domene preiskovanja sploh ne potrebuje [Sch86]. Na primer v šahu se človeški igralci navadno zanašajo na svoje znanje in izkušnje ter izvajajo manj preiskovanja, pač pa ocenjujejo nastale položaje natančno, kot je le mogoče. Popolno znanje o domeni preiskovanje naredi nepotrebno in podobno, možnost preiskovanja

vseh možnih poti ne zahteva nikakršnega domenskega znanja za uspešno rešitev problema. V praksi je značilno kombiniranje preiskovanja in uporabe domenskega znanja, saj katerikoli od obeh navedenih ekstremov navadno ni izvedljiv, vsaj pri zanimivih domenah ne [JSB⁺97].

Preiskovanje in znanje sta prav tako ključni sestavini vsakega ekspertnega sistema. Ekspertni sistemi na splošno vsebujejo funkcijo za reševanje problemov, ki uporablja značilno domensko znanje, in pri tem pogosto uporablja tudi preiskovanje [Sch86; Bra00]. Ker je znanje ključna komponenta inteligentnega računalniškega sistema, je njegovo pridobivanje ena izmed stalno prisotnih nalog v umetni inteligenci. Proces pridobivanja znanja, imenovan elicitacija znanja, je znan kot težavna naloga in predstavlja glavno ozko grlo pri gradnji baze znanja za ekspertne sisteme [Fei03].

A.1.3 Igranje iger kot platforma za raziskave v umetni inteligenci

Večine zanimivih iger ni mogoče igrati na zadovoljivi ravni brez uporabe domenskega znanja, saj je ustrezeni prostor stanj prevelik, da bi se ga dalo v celoti preiskati v razumnem času. Zato večine iger ni mogoče igrati niti izključno z zanašanjem na znanje niti izključno s preiskovanjem. Nadalje, že od samih začetkov umetne inteligence je igranje iger nudilo izvrstno platformo za izpopolnjevanje metod in algoritmov s tega področja. Pri igrah se igralci (ljudje ali računalniki) neprestano soočajo s problemi, ki jih morajo rešiti. Iz navedenih razlogov je bilo področje igranja iger izbrano kot ustrezna platforma za tematiko v tej disertaciji.

Pri igranju iger je prostor stanj navadno predstavljen kot drevo igre. V praksi računalniškega igranja iger se ustvari le del drevesa igre, drevo iskanja, in hevristična ocenjevalna funkcija oceni končne pozicije v drevesu iskanja. Hevristične ocene nekončnih pozicij pridobimo s pomočjo upoštevanja minimaks principa. Tako imenovano minimaks preiskovanje ostaja ključna komponenta programov, ki med drugim vsebujejo tudi obsežno količino znanja, pridobljenega z upoštevanjem človeških pristopov k razumevanju stanj igre in izbire potez [Bea99].

Številni eksperimenti s programi za igranje iger so bili izvedeni z namenom ugotavljanja prednosti izpopolnjenega znanja in/ali poglobljenega preiskovanja. Še posebej popularna domena v teh eksperimentih je bil šah. Eksplicitno ali implicitno sporočilo teh del je bilo, da je rezultate, pridobljene pri šahu kot domeni, mogoče posplošiti tudi na ostale igre [JS99].

A.1.4 Šah kot raziskovalna domena

Newell in Simon [NS72] postavljata šah kot posebej atraktivno raziskovalno domeno za reševanje problemov pri ljudeh iz več vzrokov.

1. Izbiri poteze pri šahu se splošno priznava status težavne naloge reševanja problemov.
2. Ogromna količina zapisanih izkušenj omogoča lažje ocenjevanje kakovosti šahovskega programa in njegovo podrobno primerjavo s človeškimi igralci z različnimi prednostmi, slogi in celo različnimi obdobji v zgodovini igre. Nadalje je mogoče protokole, ki jih naredi šahovski program, primerjati s človeškimi protokoli pri istih položajih igre.
3. Ta naloga se je že uporabljala v starejših raziskavah, predvsem v delu A. de Groota s človeškimi igralci šaha [dG78].*
4. Neenakomerna struktura šaha daje nalogi nekaj pridiha vsakdanjega, klasičnega reševanja težav, česar ne najdemo v nalogah, kot so dokazovanje teoremov ali reševanje ugank.

Schaeffer [Sch86] zagovarja, da ima šah mnoge prednosti kot domena za raziskovanje nekaterih izmed problemov v umetni inteligenci. Zmogljivost šahovskega programa je tesno povezana tako s preiskovalnimi algoritmi kot z domenskim znanjem. Zaradi kompleksnosti igre (možnih je okrog 10^{46} različnih položajev [Chi96]) je popolna igra neizvedljiva, zato morajo imeti šahovski programi splošno znanje, ki skuša opisati čim več možnih položajev. Rezultat tega je, da je znanje nenatančno (hevristično) in da mora program sprejeti pomembne odločitve glede na kakovost odziva, oziroma porabljen čas. Navede tudi nekaj drugih prednosti.

1. Igra je intelektualno zahtevna; 200 let intenzivne analize je ni uspelo narediti nič manj zanimive, niti izčrpati vseh možnosti.
2. Šahovska pravila so jasno določena.
3. Šah je mogoče razdeliti v obvladljive podskupine; na primer omejevanje področja problema na šahovske končnice.

*Tudi po letu 1978 je več drugih raziskovalcev iz področja psihologije uporabilo šah kot področje raziskovanja.)

4. Šahovsko ratingiranje je sprejeta metoda za merjenje uspešnosti. To je pomembno, saj je izboljšave v igranju šahovskega programa mogoče opaziti v njegovem ratingu.
5. Na voljo je obsežna zbirka znanja o šahu iz katere je mogoče črpati.
6. Veliko je znanega o ljudeh in računalnikih, ki igrajo šah.
7. Mnogi raziskovalci umetne inteligence znajo igrati šah in so sposobni oceniti rezultate šahovskega programa. Pri mnogih drugih aplikacijah se povprečni raziskovalec ne spozna na področje in se mora zanašati na mnenje drugih.

Sedaj, v letu 2010, lahko dodamo še naslednje trditve o primernosti šaha kot izbrane domene za naše raziskovalno delo.

1. Vse močnejši šahovski programi predstavljajo nevarne nasprotnike celo najmočnejšim šahovskim vele mojstrom, v številnih pogledih pa jih že prekašajo [Kas06].
2. Na voljo so baze podatkov, ki določajo najboljše poteze v prav vsaki poziciji [Tho86] (v vseh končnicah, kjer je prisotno 6 figur vključno s kraljema).
3. Na voljo in splošno uporabljana je uporabnikom prijazna programska oprema, ki omogoča delo z ogromnimi podatkovnimi bazami, ki vsebujejo milijone šahovskih partij.
4. Šahovski programi so sposobni sprejemati zelo kvalitetne odločitve glede najboljših potez v dani poziciji v le nekaj minutah ali celo v le nekaj sekundah. Takšni programi so enostavno dostopni in splošno uporabljani.
5. Šah je še vedno zelo priljubljena igra (ali celo še bolj kot v preteklosti).

A.1.5 Ocenjevanje in izboljševanje uspešnosti pri reševanju problemov

Osrednji dve znanstveni vprašanji v doktorski disertaciji sta:

1. Kako lahko razvijemo metode, ki bodo temeljile na računalniškem hevrističnem preiskovanju, za ocenjevanje človekove uspešnosti pri reševanju problemov?

Vprašanje vključuje:

- Kako uporabiti računalnik pri ocenjevanju uspešnosti človekovega reševanja problemov?
- Kako z modelom računalniškega reševanja oceniti, kako težavni so konkretni dani problemi za človeka?

2. Kako lahko razvijemo metode, ki bodo temeljile na računalniškem hevrističnem preiskovanju, za izboljševanje uspešnosti človeka pri reševanju problemov?

Vprašanje vključuje:

- Kako bi lahko računalniško reševanje problema uporabili za poučevanje človeka o reševanju problemov v dani problemski domeni?
- Kako transformirati znanje, izraženo v obliki, primerni za računalnik, v obliko, ki jo razume in lahko uporabi človek?

V prvem delu disertacije, "Ocenjevanje uspešnosti ljudi pri reševanju problemov", smo pokazali, da so programi, ki temeljijo na hevrističnem preiskovanju, lahko uspešni ocenjevalci uspešnosti ljudi pri reševanju problemov. Predstavili smo novo metodo, temelječo na hevrističnem preiskovanju, za ocenjevanje uspešnosti reševanja problemov v šahu (z možnostjo razširitve na ostale igre) in pokazali verodostojnost te metode. Eksperimentalni rezultati in teoretične razlage so podprle našo tezo, da lahko računalniški program z uporabo naše metode ustrezno razvrsti šahiste glede na njihovo uspešnost pri reševanju problemov, četudi je po šahovski moči slabši od njih. Prav tako smo razvili novo metodo, ki temelji na računalniškem hevrističnem preiskovanju, za ocenjevanje povprečne težavnosti množice šahovskih pozicij (problemskih situacij) za človeka.

V drugem delu disertacije, "Izboljševanje uspešnosti ljudi pri reševanju problemov", smo najprej predstavili nov pristop, temelječ na računalniškem hevrističnem

preiskovanju, k avtomatskemu in hkrati človeku razumljivemu komentiranju odločitev v šahu. Razvili smo nov pristop k formalizaciji kompleksnih vzorcev za namen računalniškega komentiranja šahovskih partij. Predstavili smo tudi nov pristop k polavtomatskemu pridobivanju človeku razumljivega znanja, primerne za poučevanje reševanja problemov v dani problemski domeni. Ustreznost tega pristopa smo preverili s študijo, kjer smo z njegovo uporabo pridobili človeku razumljiva navodila za poučevanje težavne šahovske končnice.

Tretji del disertacije, "O naravi hevrističnega preiskovanja pri računalniškem igranju iger", stremi k izboljšanju razumevanja lastnosti hevrističnega preiskovanja in posledic interakcije med znanjem in preiskovanjem, tipično prisotne pri reševanju problemov, tako pri ljudeh kot pri računalnikih. Analizirali smo lastnosti uspešnih hevrističnih ocenjevalnih funkcij pri računalniškem igranju iger. Obširneje smo raziskali lastnost *monotonosti* hevrističnih ocenjevalnih funkcij pri igranju iger in pokazali, kako lahko z upoštevanjem te lastnosti uspešno rešimo težaven tip problemov, kjer hevristično preiskovanje običajno odpove (detekcija neprebojnih *utrdb* v šahu). Razložili smo nekatere od možnih vplivov omenjene lastnosti na teorijo igranja iger. Pokazali smo tudi, da hevristične ocene, pridobljene pri različnih globinah iskanja, niso primerljive med seboj, kot je sicer splošno predpostavljeno tako v literaturi kot v praktičnih aplikacijah. V nadaljevanju smo izvedli eksperimentalno študijo v zvezi z dejavniki, ki vplivajo na spreminjanje odločitev z globino preiskovanja. Empirično smo dokazali novi ugotovitvi, da je pogostost razlik v odločitvah, ki temeljijo na različnih globinah preiskovanja, odvisna od (1) kvalitete hevrističnega znanja v ocenjevalni funkciji in (2) vrednosti vozlišča v preiskovalnem prostoru.

A.2 Ocenjevanje uspešnosti ljudi pri reševanju problemov

V prvem delu disertacije smo pokazali, da so programi, ki temeljijo na hevrističnem preiskovanju, lahko uspešni ocenjevalci uspešnosti ljudi pri reševanju problemov. Predstavili smo novo metodo, temelječo na hevrističnem preiskovanju, za ocenjevanje uspešnosti reševanja problemov v šahu (z možnostjo razširitve na ostale igre), in pokazali verodostojnost te metode. Eksperimentalni rezultati in teoretične razlage so podprle našo tezo, da lahko računalniški program z uporabo naše metode ustrezno razvrsti šahiste glede na njihovo uspešnost pri reševanju problemov, četudi je po šahovski moči slabši od njih. Prav tako smo razvili novo metodo, ki temelji na računalniškem hevrističnem preiskovanju, za ocenjevanje povprečne težavnosti množice šahovskih pozicij (problemskih situacij) za človeka.

A.2.1 Računalniška primerjava svetovnih šahovskih prvakov

Kdo je bil najboljši šahist vseh časov? To vprašanje že od nekdaj buri duhove svetovne šahovske javnosti, vendar dobro utemeljenega, objektivnega odgovora ni, saj je potrebna primerjava med igralci, ki so živeli v različnih obdobjih in se nikoli niso mogli pomeriti med seboj za šahovnico. Pojavitev vse močnejših šahovskih programov omogoča objektivnejše odgovore na to vprašanje. Vendar pa so kljub temu v dosedanjih tovrstnih raziskavah računalnike v glavnem uporabljali le za statistične obdelave *rezultatov* šahovskih partij. Le-ti pa ne izražajo vedno dejanske šahovske moči igralcev, še posebej ker je kvaliteta igre skozi desetletja na splošno močno napredovala. Jeff Sonas [Son] je leta 2005 izvedel obsežno raziskavo, ko je s pomočjo dobro premišljenega ratinškega sistema primerjal rezultatsko uspešnost igralcev vse od leta 1840 naprej, vendar je težko verjeti, da bi prvega uradnega svetovnega prvaka, Wilhelma Steinitza, ki po rezultatih Sonasove raziskave sodi med deset najboljših igralcev vseh časov, dejansko bilo mogoče uvrstiti tako visoko po njegovi šahovski moči, odraženi z njegovimi potezami na šahovnici.

Naš pristop je bil drugačen: zanimala nas je *kvaliteta igre* šahistov, ki smo jo ocenjevali s pomočjo računalniške analize posameznih *potez*. Osnovali smo tudi metodo za določanje kompleksnosti oz. težavnosti pozicij, da bi pri ocenjevanju upoštevali razlike med različnimi stili igranja. Mirni *pozicijski igralci* so namreč v svojih tipičnih partijah navadno imeli manj priložnosti za grobe taktične napake kot

taktični igralci, ki so ravno z ustvarjanjem kompleksnejših pozicij postavljali svoje nasprotnike pred nerešljive probleme, pa čeprav tudi sami pri tem niso ostali nezmotljivi. Prav tako smo podali skrbno izbrano metodologijo za uporabo računalniških šahovskih programov za ocenjevanje kvalitete igre šahistov. Gre seveda le za en (vendar zelo pomemben) vidik merjenja šahovske moči, ki ne upošteva psiholoških in ostalih dejavnikov, ki so prav tako prisotni v šahovski igri. Četudi niti računalniki niti ljudje nikoli ne bodo mogli podati dokončnega odgovora na vprašanje, kdo je bil zares najboljši šahist vseh časov (za to enostavno ni zabeleženih dovolj podatkov, na voljo imamo praktično le odigrane poteze in rezultate partij, pa tudi kriteriji za primerjavo bodo stvar diskusije), so nekatere metode vendarle bolj objektivne od drugih in računalniki kljub vsemu lahko podajo marsikatero koristne odgovore, če jih le uporabimo na primeren način.

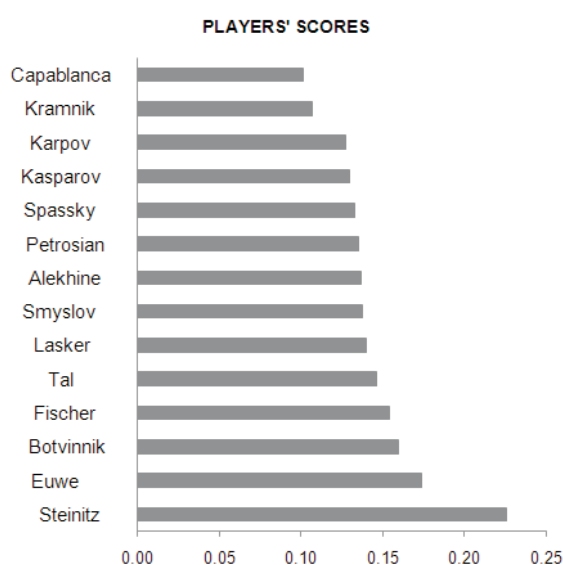
Ocenjevali smo štirinajst svetovnih prvakov, ki so vladali na svetovnem šahovskem prestolu od prvega uradnega dvoboja za svetovnega prvaka leta 1886 pa vse do leta 2006. Obravnavali smo partije njihovih neposrednih dvobojev za naslov šahovskega prvaka, bodisi so bili v njih v vlogi izzivalca bodisi v vlogi branitelja naslova. Osnovni kriterij je povprečno odstopanje med ocenami potez, ki so jih svetovni prvaki odigrali v obravnavanih partijah, in med potezami, ki jih je v istih pozicijah predlagal računalnik. Prav tako smo izračunali povprečno število grobih napak in uspešnost igralcev v primeru, da bi vsi imeli opravka z enako kompleksnimi pozicijami. Naše analize med drugim tudi jasno kažejo, da je odstotek najboljših potez odvisen od pozicije same in da je zelo močno povezan z razliko med ocenama najboljših dveh potez (kot jih predlaga računalnik): večja kot je razlika med najboljšima potezama, lažje je najti najboljšo potezo.

Ocenjevanje vsake partije se je pričelo pri 12. potezi. Današnji računalniški programi namreč slabo ocenjujejo poteze v začetni fazi igre, hkrati pa bi zgodnejši pričetek ocenjevanja partij bil po vsej verjetnosti v prid svetovnim prvakom mlajših generacij, ki so bili v fazi otvoritve zaradi stalnega napredka teorije otvoritev bolje pripravljene od njihovih predhodnikov. Vendar pa tudi ne gre prezreti dejstva, da so slednji imeli na voljo več časa za razmišljanje. Kasnejši pričetek ocenjevanja bi po drugi strani vodil do izgube koristnih informacij in 12. poteza je bila izbrana kot ustrezen kompromis.

Pri izračunu nismo upoštevali potez, pri katerih imata tako najbolje ocenjena poteza kot odigrana poteza oceno izven območja od -2.00 do +2.00. Namreč ko šahist ocenjuje, da je njegova pozicija *dobljena* oz. da ima prednost, ki zadošča, da

partijo lahko odloči v svojo korist tudi ob morebitni najboljši igri nasprotnika, včasih namenoma ne povleče najboljše poteze, ampak se odloči za potezo, ki je “dovolj dobra” in vodi do zmage z manj tveganja. Prav tako šahisti v izgubljenih pozicijah, kjer ocenijo, da se niti z morebitnimi najboljšimi potezami ne morejo rešiti poraza, če nasprotnik ne naredi kakšne večje napake, včasih namenoma povlečejo poteze, za katere sicer vedo, da so objektivno slabše, vendar jim dajejo večje praktične možnosti, da se rešijo iz težkega položaja. Neprimerno bi bilo obsojati igralce, ko so na ta povsem legitimen način poskušali priti oz. tudi prišli do zelenega rezultata.

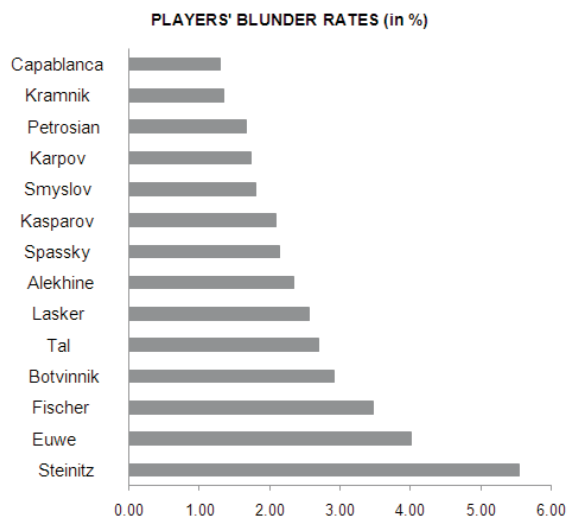
Igralec	Razlika
Capablanca	0,1012
Kramnik	0,1071
Karpov	0,1274
Kasparov	0,1297
Spasky	0,1330
Petrosian	0,1352
Alekhine	0,1368
Smyslov	0,1375
Lasker	0,1396
Tal	0,1462
Fischer	0,1541
Botvinnik	0,1595
Euwe	0,1734
Steinitz	0,2259



Slika A.1: Razvrstitev svetovnih prvakov po uspešnosti glede na osnovni kriterij, to je povprečna razlika med odigranimi in najbolj ocenjenimi potezami, uporabljen je bil računalniški program CRAFTY pri globini preiskovanja 12 polpotez.

Splošno gledano rezultati analize delujejo smiselno in se jih s strani šahovskih strokovnjakov da prav lepo interpretirati. Kljub temu so nekateri rezultati za mnoge prav gotovo lahko presenetljivi. Osnovni kriterij za ocenjevanje svetovnih prvakov je bila povprečna razlika med odigranimi in najbolj ocenjenimi potezami. Zmagovalec po tem kriteriju je tretji svetovni prvak, ki je vladal na šahovskem prestolu med leti 1921 in 1927, Jose Raul Capablanca (Slika A.1). Ta rezultat je po vsej verjetnosti povezan z relativno nizkimi izmerjenimi kompleksnostmi (oz. nižjo stopnjo težavnosti) pozicij v Capablancinih partijah (več o tem v podpoglavju A.2.3), kar se sklada s šahovsko literaturo, povezano z ocenami njegovega stila igre. Gari Kasparov v svoji zbirki knjig *Moji veliki predhodniki*, komentirajoč Capablancine partije,

Igralec	%
Capablanca	1,30
Kramnik	1,34
Petrosian	1,67
Karpov	1,73
Smyslov	1,80
Kasparov	2,09
Spassky	2,14
Alekhine	2,34
Lasker	2,56
Tal	2,96
Botvinnik	2,91
Fischer	3,47
Euwe	4,02
Steinitz	5,55



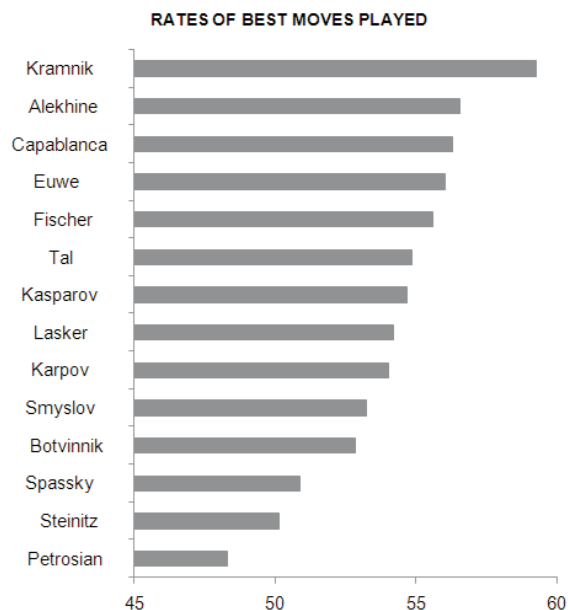
Slika A.2: Razvrstitev svetovnih prvakov po uspešnosti glede na odstotek grobih napak, uporabljen je bil računalniški program CRAFTY pri globini preiskovanja 12 polpotez.

celo navaja, da Kubanec občasno sploh ni izgubljal časa z računanjem kompleksnih taktičnih variant, ampak je preprosto dal prednost potezam, ki so pozicijsko bile tako močno utemeljene, da je bilo računanje preprosto odveč. Capablanca opisuje še z naslednjimi besedami: “Zmagoval je vse najpomembnejše partije in dvoboje ter bil neporažen več let zapored (od vseh prvakov je izgubil najmanj partij)” in “njegov stil, eden najbolj kristalno čistih v zgodovini šaha, navdušuje s svojo logiko”. [Kas06]

Rezultati merjenj grobih napak so podobni (Slika A.2). Bilo je pričakovati, da bodo pozicijski igralci v prednosti pred taktičnimi igralci, saj v mirnih pozicijah, s katerimi so se navadno soočali, ni bilo veliko možnosti za spreglede. Opaziti velja odličen rezultat Tigrana Petrosjana, ki je splošno znan kot tipičen pozicijski igralec. V skladu z omenjenim je Steinitz, ki je živel v stoletju na oko prijetnega, vendar zato manj korektnega *romantičnega šaha*, zasedel prepričljivo zadnje mesto.

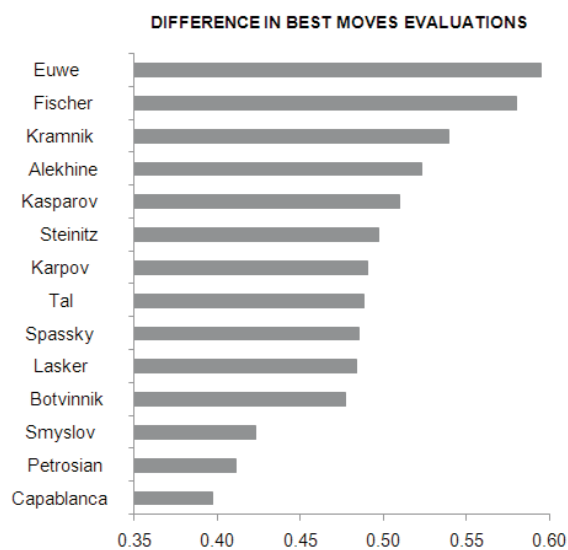
Preverili smo tudi odstotek najboljših odigranih potez glede na računalniški program CRAFTY pri globini preiskovanja 12 polpotez. Odstotek najboljših potez sam po sebi ne pove toliko o moči igralca, kot se to morda zdi na prvi pogled. V določenih tipih pozicije je veliko lažje najti najboljšo potezo kot drugje. Eksperimenti so pokazali, da je odstotek najboljših odigranih potez tesno koreliran z razliko v ocenah med najboljšima potezama v dani poziciji. Večja kot je razlika, večji je bil izmerjeni uspeh igralcev pri odločanju za najboljšo potezo.

Igralec	%
Kramnik	59,26
Alekhine	56,58
Capablanca	56,30
Euwe	56,03
Fischer	55,59
Tal	54,88
Kasparov	54,68
Lasker	54,22
Karpov	54,05
Smyslov	53,22
Botvinnik	52,83
Spassky	50,87
Steinitz	50,15
Petrosian	48,32



Slika A.3: Odstotek odigranih najboljših potez posameznih igralcev glede na računalniški program CRAFTY pri globini preiskovanja 12 polpotez.

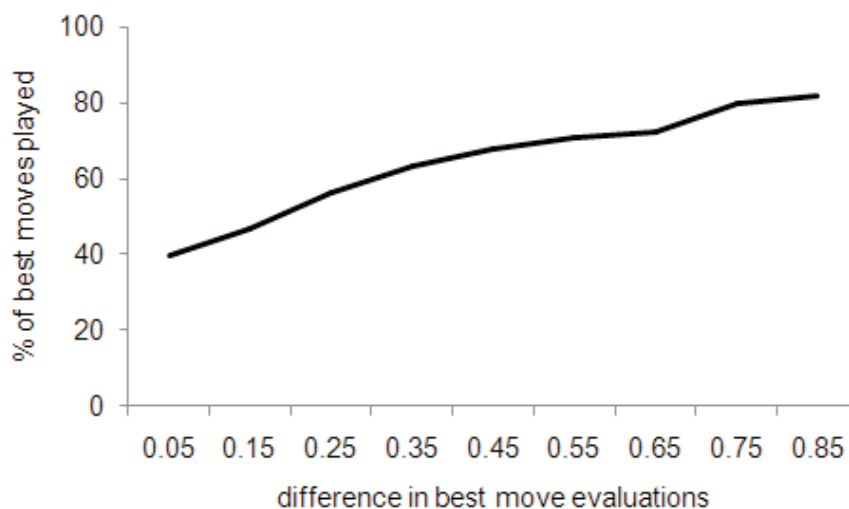
Igralec	Razlika
Euwe	0,5952
Fischer	0,5805
Kramnik	0,5397
Alekhine	0,5233
Kasparov	0,5095
Steinitz	0,4976
Karpov	0,4906
Tal	0,4887
Spassky	0,4851
Lasker	0,4841
Botvinnik	0,4770
Smyslov	0,4236
Petrosian	0,4114
Capablanca	0,3972



Slika A.4: Razlika med ocenama dveh najboljših potez glede na računalniški program CRAFTY pri globini preiskovanja 12 polpotez.

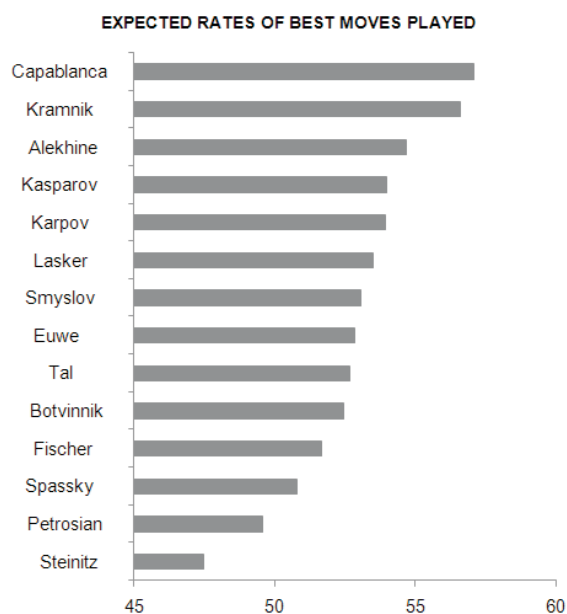
Kramnik, Fischer in Aljehin so največkrat odigrali najboljšo potezo, vendar so v njihovih pozicijah bile tudi relativno visoke razlike med najboljšima potezama (Slika A.3). V nasprotju s tem opažanjem je v pozicijah Capablance, ki po odstotku odi-

granih najboljših potez takoj sledi omenjeni trojici, bila izmerjena v povprečju najmanjša razlika med najboljšima potezama (Slika A.4).



Slika A.5: Korelacija med razliko v ocenah dveh najboljših potez in odstotkom odigranih najboljših potez.

Igralec	%
Capablanca	57,08
Kramnik	56,62
Alekhine	54,67
Kasparov	53,98
Karpov	53,94
Lasker	53,50
Smyslov	53,07
Euwe	52,08
Tal	52,66
Botvinnik	52,47
Fischer	51,68
Spassky	50,80
Petrosian	49,56
Steinitz	47,50



Slika A.6: Pričakovani odstotek odigranih najboljših potez, če bi vsi igralci imeli opravka z enakimi razlikami med najboljšima potezama.

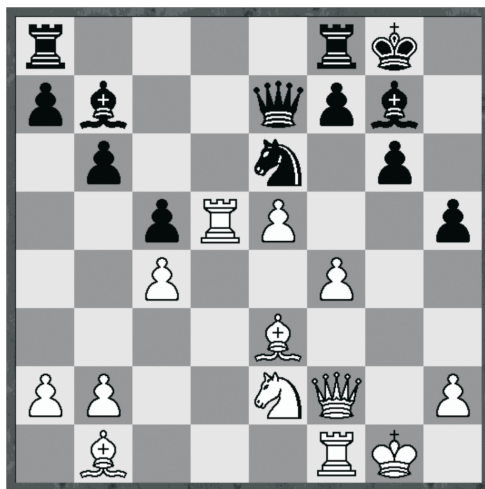
Ugotovljena korelacija (Slika A.5) je bila podlaga za oblikovanje novega kriterija: pričakovani odstotek odigranih najboljših potez, če bi vsi igralci imeli opravka z enakimi razlikami med najboljšima potezama. Ta kriterij predstavlja prvi poizkus spraviti vse prvake na "skupni imenovalec" (drugi tak poizkus je predstavljen v podpoglavju A.2.3). Zmagovalec po tem kriteriju je bil znova Capablanca. Skupaj s Kramnikom po rezultatu spet močno izstopata pred ostalimi (Slika A.6).

A.2.2 Verodostojnost ocenjevalca uspešnosti, temelječega na hevrističnem preiskovanju

Za ocenjevanje je bil uporabljen odprtokodni šahovski program CRAFTY. CRAFTY sicer ima moč šahovskega vele mojstra, kljub temu pa se poraja vprašanje, kako je mogoče za ocenjevanje uporabiti program, ki je po vsej verjetnosti slabši vsaj od večine obravnavanih svetovnih prvakov. Vendar, ocenjenih je bilo skupno več kot 37.000 pozicij in naša predpostavka je bila, da čeprav CRAFTY-jeve ocene niso vselej pravilne, morajo za tovrstno analizo biti le dovolj natančne, da se občasne manjše napake v ocenah statistično izničijo. Eksperimentalni rezultati in teoretične razlage so podprle našo tezo, da lahko računalniški program ustrezno razvrsti posamezne ljudi glede na njihovo uspešnost pri reševanju problemov, četudi je pri reševanju problemov v dani domeni slabši od njih, in da tudi uporaba katerega od močnejših šahovskih programov vodi do podobnih rezultatov. V tem razširjenem povzetku se bomo omejili le na predstavitev ključne ideje, ki je v ozadju empiričnega dokaza o verodostojnosti CRAFTY-ja kot ocenjevalca, ki temelji na hevrističnem preiskovanju.

V splošnem številni argumenti govorijo v prid računalniškemu programom kot ocenjevalcem. Pri ocenjevanju pozicij navajajo numerične ocene, ki so za naš namen neprimerno bolj uporabne kot tipične ocene ljudi šahistov, hkrati pa se pri ocenjevanju ves čas držijo istih pravil in so zato pri ocenah veliko bolj konsistentni od ljudi in tudi niso pod vplivom čustev in drugih, za ocenjevanje motečih dejavnikov, ter so povrh vsega izredno dobri pri detektiranju taktičnih napak. Poleg tega je bil CRAFTY pri ocenjevanju omejen na fiksno globino preiskovanja (in ne časovno), kar pomeni, da je bolj težavnim pozicijam avtomatsko namenjal več časa.

Vendar šahovski programi podajajo različne ocene za iste ocenjevane pozicije in splošna intuicija nas kaj lahko vodi do zaključka, da bi ocenjevanje z različnimi programi dalo povsem različne rezultate. Še več, celo programi sami tipično spreminjajo svoje odločitve z naraščajočo globino preiskovanja. Kot tabela na Sliki A.7 nazorno



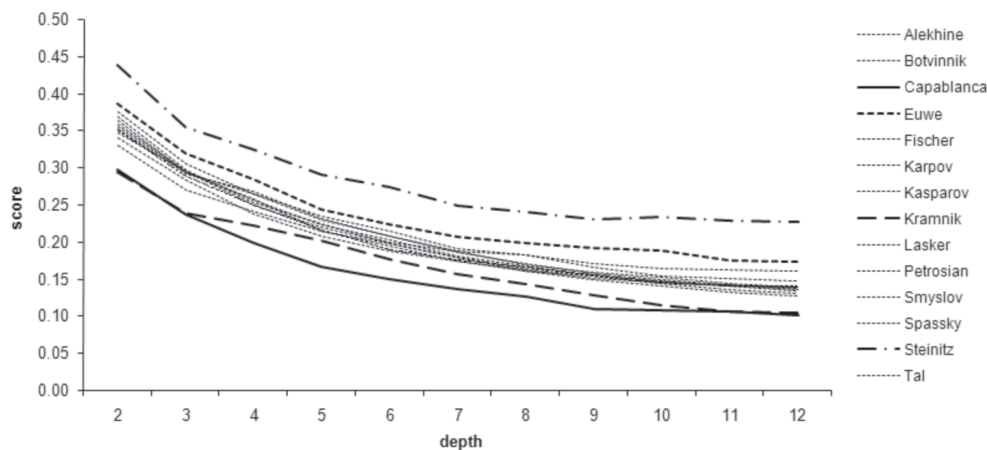
Globina	Poteza	Ocena
2	Lxd5	-1,46
3	Lxd5	-1,44
4	Lxd5	-0,75
5	Lxd5	-1,00
6	Lxd5	-0,60
7	Lxd5	-0,76
8	Tad8	-0,26
9	Lxd5	-0,48
10	Tfe8	-0,14
11	Lxd5	-0,35
12	Sc7	-0,07

Slika A.7: Botvinnik-Talj, 17.partija za naslov svetovnega prvaka, Moskva 1961. V poziciji na diagramu je Talj odigral potezo 23...Sc7! in kasneje, v 83. potezi, tudi zmagal to partijo. Tabela na desni prikazuje CRAFTY-jeve odločitve glede najboljših potez in njegove ocene te pozicije pri različnih globinah preiskovanja.

prikazuje, CRAFTY pri različnih globinah iskanja pozicijo na diagramu ocenjuje zelo različno: ne samo, da se izrazito menjajo ocene pozicije, tudi sama izbira najboljših potez je od globine preiskovanja 7 polpotez dalje vselej različna. Podobno velja za računalniške šahovske programe tudi na splošno: ocene in predlagane najboljši poteze se pogosto spreminjajo z globino preiskovanja.

Pojavi se vprašanje: “Kako bi CRAFTY razvrstil ocenjevane svetovne prvake, če bi ga omejili na različne globine preiskovanja?” Glede na omenjene razlike v ocenah pri različnih globinah nam intuicija sugerira na videz jasen odgovor: pri vsaki globini bi dobili povsem različne rezultate. Vendar pa so naše raziskave pokazale, da bi CRAFTY tudi pri preiskovanju do različnih globin v dobršni meri ohranil razvrstitev igralcev. Npr. prav pri vseh globinah sta Capablanca in Kramnik na prvih dveh mestih, Euwe in Steinitz pa na zadnjih dveh (Slika A.8). Pri nekaterih igralcih vmes se vrstni red sicer spreminja, kar pa je posledica dejstva, da so njihovi rezultati zelo izenačeni.

Kaj ta rezultat pomeni? Leta 1982 je Ken Thompson [Tho82] primerjal med seboj programe pri različnih globinah preiskovanja in pokazal, da je program, ki išče samo eno polpotezo globlje, boljši za več kot 200 ratinških točk od enakega programa, ki preiskuje do la za eno polpotezo nižje globine. Naši rezultati ocenjevanja s CRAFTY-jem do različnih globin torej kažejo na to, da čeprav bi uporabili različno močne



Slika A.8: Razvrstitev svetovnih prvakov po uspešnosti glede na osnovni kriterij, pri različnih globinah preiskovanja.

verzije CRAFTY-ja, bi še vedno dobili praktično enako razvrstitev igralcev. Povsem umestno je tudi sklepanje, da bi CRAFTY na zelo podoben način ohranil vrstni red igralcev tudi pri globinah iskanja do 20 in še dlje. CRAFTY pri tako velikih globinah iskanja pa bi se po vsej verjetnosti lahko povsem enakomerno kosal z najboljšimi računalniškimi šahovskimi programi, ki bi preiskovali do nekoliko nižjih globin.

A.2.3 Ocenjevanje težavnosti šahovskih pozicij

Metode hevrističnega reševanja problemov so tudi dober model človeškega reševanja problemov, ki prav tako odraža preiskovanje in znanje o konkretnem problemu. Seveda pa sta pri računalnikih in ljudeh ti dve komponenti zelo različno zastopani. Človek – ekspert tipično uporablja veliko bogatejše znanje o samem problemu, medtem ko je prednost računalnika v neprimerno hitrejšem preiskovanju. Pionirsko delo o modeliranju človeškega reševanja problemov z računalniškim sta opravila v svoji dobro znani raziskavi A. Newell in H.A. Simon [NS72].

Čeprav je glede količine preiskovanja pri reševanju problemov med računalnikom in človekom razlika ogromna, kljub temu obstajajo določene podobnosti med njunima načinoma iskanja najboljše poteze v dani poziciji. Tako računalnik kot človek imata opravka z gigantskim drevesom iskanja, kjer koren predstavlja trenutna pozi-

cija, ki je predmet reševanja problemov, liste tega drevesa pa predstavljajo vse možne poteze oz. pozicije, do katerih te poteze vodijo – in tako rekurzivno naprej iz vsakega možnega vozlišča. Oba poizkušata najti najboljša nadaljevanja in se pri tem trudita ne upoštevati potez, ki niso relevantna za oceno možnih nadaljevanj. Razlikujeta se predvsem v načinu, kako ne upoštevati nerelevantnih nadaljevanj. Računalniki se pri tem zanašajo na algoritme za učinkovito rezanje poddreves, medtem ko se človek pri tem zanaša na svoje znanje in izkušnje. Oba sta pri preiskovanju omejena s časom, zato ne moreta preiskovati do poljubne globine, pač pa morata v določenem trenutku oceniti dano pozicijo. Oba se pri tem poslužujeta delnih ocen. Medtem ko računalnik uporablja numerične hevristične ocene, ima človek navadno v mislih deskriptivno oceno, kot so npr. “majhna prednost”, “odločilna prednost” ali “nejasna pozicija”. Tako računalnik kot človek preverjata vse t.i. *forsirane* variante – računalnik v ta namen tem uporablja *iskanje mirovanja* (ang. *quiescence search*) – preden ocenita pozicijo v korenu drevesa. Možno je torej najti precej vzporednic med računalniškimi in človeškimi postopki za iskanje najboljše poteze, kar je služilo kot osnova za osnove metode za ocenjevanje težavnosti (oz. kompleksnosti) šahovskih pozicij.

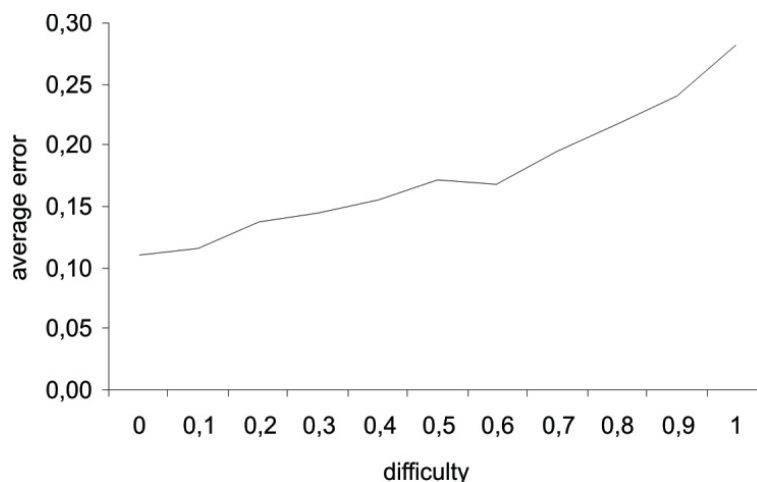
Osnovna ideja je sledeča: dana pozicija je težavna za dodelitev točne ocene in izbire najboljšega nadaljevanja, ko se pri različnih globinah preiskovanja pojavijo različne kandidatske poteze, ki signifikantno spreminjajo oceno trenutne pozicije v korenu. V takih primerih mora igralec preanalizirati več možnih nadaljevanj in preiskovati do večje globine, da najde poteze, ki vplivajo na oceno pozicije v korenu in nato izbere najboljšo izmed njih. Realizacijo te ideje v obliko, primerno za uporabo v računalniškem programu, ilustrira Algoritem 2. Naj poudarimo, da namen te metrike ni odražati resnične težavnosti šahovskih pozicij s kognitivnega vidika, niti ni osnovana za primerjanje *posameznih* šahovskih pozicij v smislu težavnosti za reševanje ustreznih problemov, ki se pri njih pojavljajo. Namen te metrike je omogočiti ocenjevanje povprečne težavnosti pri večjih količinah šahovskih pozicij, kot je bilo o mogoče v primeru analize dvobojev svetovnih prvakov.

Graf odvisnosti napak svetovnih prvakov glede na različne nivoje težavnosti pozicij (Slika A.9) jasno kaže na korektnost uporabljene metode za merjenje težavnosti pozicij. Igralci so delali manj napak v bolj preprostih pozicijah, povprečne izmerjene napake pa naraščajo s težavnostjo pozicij.

Capablanca je znan po tem, da je igral relativno “enostaven” šah in se je izogibal komplikacijam, medtem ko sta se Steinitz in Talj v svojih partijah pogosto soočala z “divjimi” pozicijami. Rezultati merjenja težavnosti pozicij (Slika A.10) se povsem

Algorithm 2 Algoritem za izračun težavnosti pozicije.

```
težavnost := 0;
for globina 2 to 12 do
  if (globina > 2) and (prej_najboljša_poteza not equals zdaj_najboljša_poteza)
  then
    težavnost += |ocena_najboljše_poteze - ocena_druga_najboljše_poteze|
  end if
end for
```



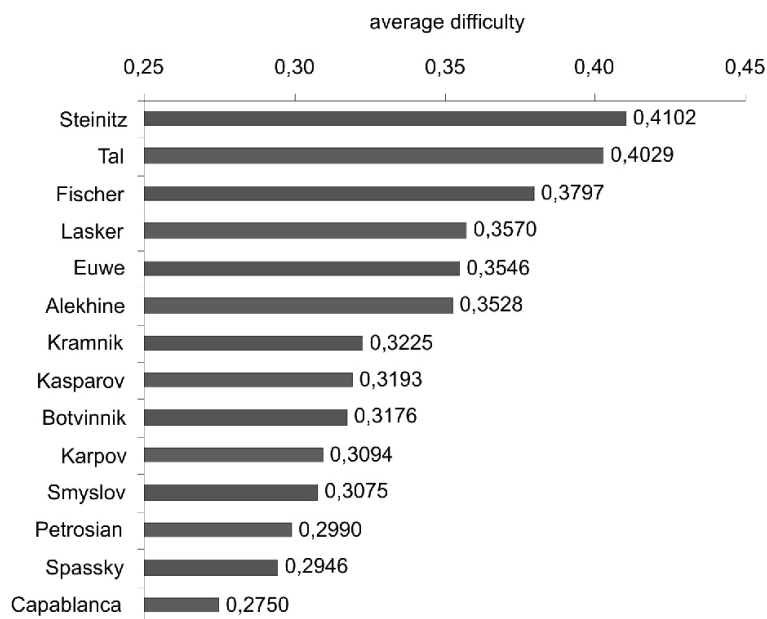
Slika A.9: Povprečna razlika med odigranimi in najbolj ocenjenimi potezami (povprečna napaka) pri različnih nivojih težavnosti.

skladajo s tem v šahovskem svetu priznanim splošnim mnenjem.

Merilo za ugotavljanje težavnosti pozicij smo uporabili, da bi ugotovili porazdelitev potez po posameznih intervalih kompleksnosti, kar je v veliki meri povezano s *stilom igre* igralcev. Capablanca, na primer, je glede na rezultate merjenja težavnosti po izbrani metodi imel veliko manj pogosto opravka z bolj težavnimi pozicijami v primerjavi s Taljem, ki velja za taktičnega igralca.

Glavna slabost uporabljenega osnovnega kriterija ocenjevanja uspešnosti (povprečna razlika med odigranimi in najbolj ocenjenimi potezami) je, kot smo že omenili, da je nekoliko nepošten do taktičnih igralcev, ki so v povprečju imeli opravka z bolj težavnimi pozicijami, vendar so se v njih tudi bolje znašli od svojih nasprotnikov in so izkoriščali to dejstvo za doseganje odličnih rezultatov. Kriterij za določanje težavnosti pozicij nam je omogočil eksperiment, v katerem smo preverili, kako bi se igralci odrezali, če bi vsi imeli opravka z enako kompleksnimi pozicijami: npr. če bi vsi igrali v stilu Capablanke ali Talja.

Videti je, da naša metoda ugotavljanja težavnosti pozicij daje smiselne rezul-



Slika A.10: Rezultati merjenja povprečne težavnosti pozicij, s katerimi so se soočali posamezni igralci v dvobojih za naslov svetovnega prvaka.

tate, ki se kvalitativno ujemajo z opažanji, ki jih v obliki komentarjev zasledimo pri šahovskih ekspertih v zvezi s stili igranja svetovnih prvakov. Kot možno nadaljevanje tega dela bi bilo zanimivo izvesti psihološko študijo z namenom ugotoviti, kako dobro naša metoda odraža resnično kognitivno težavnost šahovskih pozicij.

A.3 Izboljševanje uspešnosti ljudi pri reševanju problemov

V drugem delu disertacije smo najprej predstavili nov pristop, temelječ na računalniškem hevrističnem preiskovanju, k avtomatskemu in hkrati človeku razumljivemu komentiranju odločitev v šahu. Razvili smo nov pristop k formalizaciji kompleksnih vzorcev za namen računalniškega komentiranja šahovskih partij. Predstavili smo tudi nov pristop k polavtomatskemu pridobivanju človeku razumljivega znanja, primerne za poučevanje reševanja problemov v dani problemski domeni. Ustreznost tega pristopa smo preverili s študijo, pri kateri smo z njegovo uporabo pridobili človeku razumljiva navodila za poučevanje težavne šahovske končnice.

A.3.1 Avtomatsko komentiranje šahovskih partij

Dandanašnji šahovski programi se enakovredno kosajo s človeškimi vele mojstri, v številnih pogledih pa jih tudi že prekašajo. Kljub temu so njihove sposobnosti razložiti v ljudem razumljivem jeziku, zakaj so določene poteze dobre in zakaj ne, zelo omejene. Avtomatskemu inteligentnemu komentiranju šahovskih partij je bilo posvečeno le malo pozornosti, napredek na tem področju pa je zanemarljiv v primerjavi z gromozanskim skokom v šahovski moči programov, ki smo mu bili priče v zadnjih desetletjih. Tipični "komentariji" v obliki najboljših nadaljevanj in njihovih numeričnih ocen so le stežka v pomoč šahistu, ki bi se rad naučil pomembnih konceptov, ki se skrivajo za izbranimi potezami.

Ideja o avtomatskem komentiranju šahovskih partij ni nova. Verjetno prvi, ki je predlagal raziskave v tej smeri, je bil leta 1980 Donald Michie. Revija ICGA je kmalu nato začela podeljevati letne t.i. Herschbergove nagrade za najboljši šahovski komentatorski program [LM96]. Komentariji zmagovalcev kljub temu še do danes ostajajo zelo skopi ter so pretežno taktične narave, medtem ko kompleksnejši strateški koncepti in plani ostajajo bolj ali manj neomenjeni. Znanstvene raziskave s tega področja so večinoma omejene le na končnice šahovskih partij, predstavljeni koncepti pa imajo skupno slabost - nezmožnost praktične razširitve komentiranja na celotno šahovsko partijo [GGM93; Sei94; HB96].

Z razvojem inteligentnega računalniškega sistema, ki bo na človeku razumljiv, prijazen in zanimiv način komentiral šahovske poteze, bi radi šahovsko moč, ki jo demonstrirajo današnji programi za igranje šaha, izkoristili za poučevanje šaha ter za komentiranje šahovskih partij. Glavna prednost našega pristopa je, da omogoča komentiranje šahovskih partij v vseh fazah igre, hkrati pa avtomatsko generirani komentariji poleg zmožnosti komentiranja taktičnih pozicij izražajo tudi razumevanje strateških nians v pozicijah.

Na poti do cilja nas čakajo še prenekateri izzivi. Poleg predstavljenih težav pri uporabi strojnega učenja pri razvoju ekspertnega sistema je potrebno rešiti še nekatere probleme kognitivne narave, npr. kdaj komentirati in kdaj ne, kako dolge variante podajati, podajati komentarje za uporabnike z različnim šahovskim predznanjem itd. Eden izmed ciljev, ki jih želimo doseči, je uporabiti naš inteligentni sistem tudi za avtomatsko generiranje povzetkov šahovskih partij.

A.3.2 Formalizacija kompleksnih vzorcev za namen komentiranja odločitev pri reševanju problemov

Omogočanje naprednejših komentarjev v zvezi z odločitvami pri reševanju problemov zahteva zajem znanja za konstrukcijo vzorcev bolj kompleksne narave. V ta namen smo razvili nov pristop k formalizaciji kompleksnih vzorcev za komentiranje odločitev pri reševanju problemov v šahu. Naš pristop temelji na argumentiranem strojnem učenju [MvB07]. Raziskali smo naslednji vidik razvoja inteligentnega sistema za komentiranje šahovskih partij: možnost podajanja inteligentnih *pozicijskih* komentarjev (za razliko od komentarjev *taktične* narave). Ta naloga je toliko težja, ker moč dandanašnjih šahovskih programov izhaja predvsem iz preiskovanja in ne toliko iz znanja o številnih niansah strateške narave, ki je potrebno za uspešno komentiranje pozicijskih elementov šahovske igre. Komponente ocenjevalnih funkcij računalniških programov zato niso zadostne za naprednejše pozicijske komentarje. Formalizacija globokih pozicijskih vzorcev zahteva močna orodja za zajemanje ekspertnega znanja. Naša študija, kjer smo formalizirali koncept slabega lovca v šahu, kaže na to, da argumentirano strojno učenje predstavlja ustrezno metodo za ta namen.

A.3.3 Polavtomatsko sintetiziranje človeku razumljivega znanja iz tabeliranih šahovskih baz

V šahu so na voljo tabelirane baze podatkov (ang. *tablebases*), ki vsebujejo popolno znanje v smislu, da podajajo najboljše poteze v prav vsaki poziciji. Tovrstne baze obstajajo za vse končnice, kjer je prisotno največ 6 figur vključno s kraljema. Računalnikom te baze omogočajo optimalno igro v smislu doseganja mata v najmanjšem možnem številu potez. Vendar pa je znanje v dani obliki skoraj povsem neprimerno za človeka, ki bi se iz teh baz rad naučil pomembnih konceptov in strategij.

Razvili smo nov pristop k polavtomatskemu sintetiziranju človeku razumljivega znanja, primerne za poučevanje kako reševati probleme v dani problemski domeni. Ta pristop omogoča formalizacijo konceptov in strategij, ki se jih da uporabiti za konstrukcijo hevristične ocenjevalne funkcije, s pomočjo katere je mogoče komentirati odločitve pri reševanju problemov. V naši študiji smo polavtomatsko sintetizirali navodila za poučevanje težavne šahovske končnice matiranja z lovцем in konjem. Nadalje, pridobljena ciljno orientirana pravila smo uporabili v hevristični ocenje-

valni funkciji, s pomočjo katere smo ustvarili primere partij z avtomatsko generiranimi navodili. Formalizirana strategija je bila spoznana za primerno za namene poučevanja s strani šahovskih učiteljev. Eden od navedenih razlogov, ki podpirajo to oceno, je bil, da navodila “jasno demonstrirajo vmesne cilje na poti do matiranja”.

Naš postopek k polavtomatski sintezi znanja kombinira ideje iz argumentiranega strojnega učenja s specializiranim minimaks preiskovanjem. Z njegovo uporabo domenski ekspert in metoda strojnega učenja iterativno izboljšujeta model pravil, ki predstavljajo formalizirano strategijo. S stališča domenskega eksperta so še posebej primerne naslednje prednosti, ki jih argumentirano strojno učenje ponuja:

- ekspertom je lažje argumentirati specifične primere kot podajati splošno domensko znanje,
- ekspertom olajša prilagajanje nivoja novo vpeljanih konceptov nivoju ciljne publike študentov,
- ekspertom je potrebno podajati le znanje, ki je relevantno, in
- pridobljeno znanje je:
 - konsistentno z ekspertnim znanjem
 - v obliki, primerno za uporabo v računalniških programih za poučevanje,
 - v obliki, ki jo razume in lahko uporabi človek.

Razložili smo smernice za interakcijo med računalnikom in ekspertom za pridobivanje človeku razumljivega, ciljno orientiranega modela pravil za poučevanje, kako reševati probleme v dani problemski domeni. Pojasnili smo tudi, kako se navodila, vključno z ilustrativnimi diagrami, pridobijo polavtomatsko iz takšnega modela.

A.4 O naravi hevrističnega preiskovanja pri računalniškem igranju iger

Tretji del disertacije, stremi k izboljšanju razumevanja lastnosti hevrističnega preiskovanja in posledic interakcije med znanjem in preiskovanjem, tipično prisotne pri reševanju problemov, tako pri ljudeh kot pri računalnikih. Analizirali smo lastnosti

uspešnih hevrističnih ocenjevalnih funkcij pri računalniškem igranju iger. Podrobneje smo raziskali lastnost *monotonosti* hevrističnih ocenjevalnih funkcij pri igranju iger in pokazali, kako lahko z upoštevanjem te lastnosti uspešno rešimo težaven tip problemov, kjer hevristično preiskovanje običajno odpove (detekcija neprebojnih *utrdb* v šahu). Razložili smo nekatere od možnih vplivov nove ugotovitve na teorijo igranja iger. Pokazali smo tudi, da hevristične ocene, pridobljene pri različnih globinah iskanja, niso primerljive med seboj, kot je sicer splošno predpostavljeno tako v literaturi kot v praktičnih aplikacijah. V nadaljevanju smo izvedli eksperimentalno študijo v zvezi z dejavniki, ki vplivajo na spreminjanje odločitev z globino preiskovanja. Empirično smo dokazali novi ugotovitvi, da je pogostost razlik v odločitvah, ki temeljijo na različnih globinah preiskovanja, odvisna od (1) kvalitete hevrističnega znanja v ocenjevalni funkciji in (2) vrednosti vozlišča v preiskovalnem prostoru.

A.4.1 Monotonost kot lastnost hevrističnih ocenjevalnih funkcij

V tem poglavju smo analizirali lastnosti uspešnih hevrističnih ocenjevalnih funkcij pri igranju iger. Izbrali smo več sodobnih šahovskih programov, ki so izvajali hevristično preiskovanje iz velikega števila (več deset tisoč) šahovskih pozicij iz resničnih partij. Iz vsake pozicije se je izvajalo preiskovanje do globin v razponu od *min* do *max* polpotez. Preiskovanje do globine *d* tukaj pomeni *d* polpotez, razširjeno z iskanjem mirovanja (ang. *quiescence search*), za zagotavljanje ocenjevanja le stabilnih pozicij. Pozicije iz zgodnje faze partije niso bile predmet analize, saj sodobni šahovski programi relativno slabo ocenjujejo tovrstne pozicije (pri igranju otvoritvenih potez uporabljajo otvoritveno knjižico). Pri šahovskih programih so pridobljene ocene tipično v numerični obliki, kar je primerno za naš namen. Rezultate preiskovanja smo shranili v relacijsko podatkovno bazo, skupaj s podatki o vsaki analizirani partiji.

Na podlagi dognanj iz eksperimentalnih rezultatov smo predstavili lastnost *monotonosti* hevrističnih ocenjevalnih funkcij pri igranju iger. In sicer: z naraščajočo globino preiskovanja morajo vzratne ocene vozlišč težiti k monotonemu približevanju končnim vrednostim v prostoru preiskovanja. Eksperimentalni rezultati so pokazali, da ocenjevalne funkcije tipičnih šahovskih programov imajo to lastnost, ki programe usmerja k doseganju zelenega cilja. Vzratne ocene vozlišč torej ne aproksimirajo nekih “resničnih” vrednosti ali “idealnih” hevrističnih vrednosti, kar se sicer v literaturi na splošno predpostavlja, kar pa pomeni, da uspešne hevristične ocenje-

valne funkcije oz. njihove vzratne ocene *ne* smejo upoštevati minimaks relacije. To je, vzratne hevristične vrednosti ne smejo biti nespremenljive tekom drevesa igre, kakor so teoretične ocene igre v teoretičnem minimaks modelu.

Izvedli smo praktično demonstracijo upoštevanja v hevrističnih ocenjevalnih funkcijah prisotne lastnosti monotonosti za uspešno reševanje težavnega tipa problemov, kjer hevristično preiskovanje običajno odpove: detekcijo neprebojnih utrd (ang. *fortress*) v šahu. To so pozicije, kjer ima navadno določena stran večjo materialno prednost, vendar zmage ni mogoče doseči, saj se je nemogoče uspešno prebiti v tabor nasprotnika. Programi takšne pozicije ocenjujejo kot dobljene za močnejšo stran, kar pomeni, da so pripravljene zaiti v tovrstne pozicije, četudi bi imeli na voljo boljše nadaljevanje (četudi ocenjeno z nižjimi hevrističnimi vrednostmi). Skladno z našimi predpostavkami so eksperimenti pokazali, da v tovrstnih pozicijah kljub visokim ocenam ni sicer pričakovanega monotonega naraščanja ocen z globino preiskovanja.

Razložili smo nekatere od možnih vplivov ugotovljene lastnosti na teorijo igranja iger, kot sta (1) njen vpliv na spreminjanje odločitev z naraščajočo globino in (2) primerljivost ocen pri različnih globinah preiskovanja. Namreč, pri vozliščih z zelo visoko ali zelo nizko vzratno hevristično vrednostjo, pri katerih ocene z naraščajočo globino preiskovanja hitreje naraščajo oz. padajo, je verjetnost spreminjanja odločitev z naraščajočo globino manjša kot v vozliščih s povprečno vrednostjo. In nadalje, zaradi lastnosti monotonosti hevristične ocene, pridobljene pri različnih globinah iskanja, ne morejo biti primerljive med seboj, kar je sicer splošno (in narobe) predpostavljeno tako v literaturi kot v praktičnih aplikacijah.

A.4.2 Dejavniki, ki vplivajo na spreminjanje odločitev z globino preiskovanja

V okviru prizadevanja k izboljšanju splošnega razumevanja lastnosti hevrističnega preiskovanja smo obravnavali tudi možne dejavnike, ki vplivajo na spreminjanje odločitev z naraščajočo globino preiskovanja. Obnašanje računalniških programov v smislu spreminjanja odločitev z naraščajočo globino preiskovanja je bilo predmet številnih raziskav, še posebej glede ugotavljanja morebitnega neenakomernega oz. pojenjajočega naraščanju moči programa z vsako nadaljnjo globino (ang. *diminishing returns*). Rezultati teh raziskav so bili zelo spremenljivi glede na uporabljene eksperimentalne podatke, vzroki za to pa niso nikoli bili pojasnjeni.

Izvedli smo empirično študijo, v kateri smo uporabili bistveno večjo količino po-

datkov (preko 40.000 analiziranih pozicij iz šahovskih partij) kot ostali raziskovalci, hkrati pa smo te pozicije preanalizirali s tremi različnimi šahovskimi programi, ki so v ta namen izvajali hevristično preiskovanje iz danih pozicij (oz. problemskih situacij iz resničnih partij) do različnih globin (od 2 do 12 polpotez globoko). Podatke smo za namene te študije razdelili na več podmnožic, glede na pričakovano vrednost pozicij v njih. Ker je dobro znano, da moč šahovskih programov narašča z globino preiskovanja, so ocene, pridobljene pri največji globini preiskovanja, služile kot najbolj zanesljivo orodje za takšno razdelitev. Spreminjanja odločitev z globino smo opazovali pri vsaki podmnožici posebej. Uporabili smo programe z različno kvalitetnimi ocenjevalnimi funkcijami.

Empirično smo dokazali novi ugotovitvi, da je pogostost razlik v odločitvah, ki temeljijo na različnih globinah preiskovanja, odvisna od

- kvalitete hevrističnega znanja v ocenjevalni funkciji in
- dejanske vrednosti (glede na fiksno globino preiskovanja) vozlišča v preiskovalnem prostoru.

Pri programih s kvalitetnejšo ocenjevalno funkcijo lahko pričakujemo manj odločitev z naraščajočo globino preiskovanja (v primeru "idealne" hevristične ocenjevalne funkcije, ki bi vsebovala popolno znanje o domeni, sprememb v odločitvah pri nadaljnjih globinah preiskovanja sploh ne bi bilo). Hkrati pa se izkaže, kot to smo razložili pri ugotavljanju lastnosti monotonosti pri hevrističnih ocenjevalnih funkcijah, da je v dobljenih in izgubljenih pozicijah tovrstnih sprememb v odločitvah opazno manj kot v izenačenih pozicijah.

Pokazali smo tudi, z visoko stopnjo statistične značilnosti, da spreminjanje odločitev pojenja z naraščajočo globino preiskovanja.

A.5 Prispevki k znanosti

Prispevki disertacije spadajo v področje umetne inteligence. Raziskave, ki so vodile do teh prispevkov, so bile opravljene v ogrodju človeškega in računalniškega igranja iger, kot raziskovalna domena pa je bil uporabljen šah. Številni raziskovalci so uporabljali igranje iger kot platformo za svoje raziskave, šah pa je bil še posebej popularna domena. Eksplicitno ali implicitno sporočilo njihovih del je bilo, da je rezultate, pridobljene v domeni šaha, mogoče posplošiti tudi na ostale domene. Čeprav naše delo ne vsebuje eksplicitnih dokazov, ki bi podprli to trditev, verjamemo, da spodaj naštetih prispevkov k znanosti imajo potencial širitve tako na številne ostale igre kot tudi na nekatere druge domene, kjer je smiselno uporabiti hevristično preiskovanje.

Glavni prispevki disertacije so:

1. Razvili smo novo metodo, ki temelji na računalniškem hevrističnem preiskovanju, za ocenjevanje *človekove uspešnosti* pri reševanju problemov in utemeljili verodostojnost te metode ocenjevanja.
2. Razvili smo novo metodo, ki temelji na računalniškem hevrističnem preiskovanju, za ocenjevanje *težavnosti* danih problemov za človeka.
3. Razvili smo nov pristop k avtomatskemu, človeku razumljivemu komentiranju odločitev pri reševanju problemov, ki temelji na hevrističnem preiskovanju.
4. Razvili smo nov pristop k formalizaciji kompleksnih vzorcev za namen komentiranja odločitev pri reševanju problemov in/ali poučevanja.
5. Razvili smo nov pristop za polavtomatsko sintetiziranje človeku razumljivega znanja, primerne za poučevanje kako reševati probleme v dani problemski domeni.
6. Obširna raziskava lastnosti monotonosti uspešnih hevrističnih ocenjevalnih funkcij: z naraščajočo globino preiskovanja morajo vzvratne ocene vozlišč težiti k monotonemu približevanju končnim vrednostim v prostoru preiskovanja. Pokazali smo, da vzvratne ocene ne aproksimirajo nekaterih "resničnih" vrednosti ali "idealnih" hevrističnih vrednosti, kar se sicer v literaturi na splošno predpostavlja, in da uspešne hevristične ocenjevalne funkcije ne smejo upoštevati minimaks relacije. To je, vzvratne hevristične vrednosti ne smejo biti nespre-

menljive tekom drevesa igre, kakor so teoretične ocene igre v teoretičnem min-
imaks modelu.

7. Empirično smo dokazali novi ugotovitvi, da je pogostost razlik v odločitvah,
ki temeljijo na različnih globinah preiskovanja, odvisna od:

- kvalitete hevrističnega znanja v ocenjevalni funkciji,
- dejanske vrednosti (glede na fiksno globino preiskovanja) vozlišča v pre-
iskovalnem prostoru.

Bibliography

- [ABB⁺04] J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.
- [ABCL90] J.R. Anderson, F. Boyle, A.T. Corbett, and M.W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42(1):7–49, 1990.
- [Abr89] B. Abramson. Control strategies for two-player games. *ACM Computing Surveys*, 21(2):137–161, 1989.
- [ACKP95] J.R. Anderson, A.T. Corbett, K.R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4(2):167–207, 1995.
- [AK90] J.R. Anderson and N. Kushmerick. A rational analysis of production system architecture. *Bulletin of the Psychonomic Society*, 28(6):509, 1990.
- [And93a] J.R. Anderson. Problem solving and learning. *American Psychologist*, 48(1):35–44, 1993.
- [And93b] J.R. Anderson. *Rules of the Mind*. Erlbaum, Hillsdale, NJ, 1993.
- [Ant08] C. Antunes. Acquiring background knowledge for intelligent tutoring systems. In *EDM 2008*, pages 18–27, 2008.
- [AP91] J.R. Anderson and R. Pelletier. A developmental system for model-tracing tutors. In L. Bimbaum, editor, *The International Conference on the Learning Sciences*, pages 1–8, 1991.
- [Bea99] D.F. Beal. *The Nature of Minimax Search*. PhD thesis, IKAT, Universiteit Maastricht, 1999.
- [BH95] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.

- [Blo84] B. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.
- [Bra77] M.A. Bramer. *Representation of knowledge for chess endgames: towards a self-improving system*. PhD thesis, The Open University: Faculty of Mathematics, Milton Keynes, England, 1977.
- [Bra82] I. Bratko. Knowledge-based problem-solving in AL3. *Machine Intelligence*, 10:73–100, 1982.
- [Bra00] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, 2000.
- [Cam88] D. Campbell. Task complexity: A review and analysis. *Academy of Management Review*, 13(1):40–52, 1988.
- [CB91] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *EWSL*, pages 151–163, 1991.
- [Che06a] Chessbase.com. Computer analysis of world champions. <http://www.chessbase.com/newsdetail.asp?newsid=3465>, 2006.
- [Che06b] Chessbase.com. Computers choose: who was the strongest player? <http://www.chessbase.com/newsdetail.asp?newsid=3455>, 2006.
- [Chi96] S. Chinchalkar. An upper bound for the number of reachable positions. *ICGA Journal*, 19(3):181–183, 1996.
- [Cla87] W.J. Clancey. *Knowledge-based Tutoring: The GUIDON Program*. MIT Press, Cambridge, MA, USA, 1987.
- [Cou07] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*. Springer, 2007.
- [Del23] D. Delétang. Mat avec le fou et le cavalier. *Las Stratégie*, 56(2):25–32, 1923.
- [dG78] A.D. de Groot. *Thought and choice in chess*. Mouton Publishers, The Hague, Netherlands, 1978.
- [DvdHU05] H.H.L.M. Donkers, H.J. van den Herik, and J.W.H.M. Uiterwijk. Selecting evaluation functions in opponent-model search. *Theoretical Computer Science*, 349(2):245–267, 2005.

- [Dvo08] M. Dvoretzky. *Dvoretzky's Endgame Manual, 2nd edition*. Russell Enterprises, Inc., 2008.
- [FÖ1] J. Fürnkranz. Machine learning in games: A survey. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*. New York, NJ: Nova Scientific Publishers, 2001.
- [Fei03] E.A. Feigenbaum. Some challenges and grand challenges for computational intelligence. *Source Journal of the ACM*, 50(1):32–40, 2003.
- [FR86] R. Forsyth and R. Rada. *Machine learning applications in expert systems and information retrieval*. Halsted Press, New York, 1986.
- [GB06] M. Guid and I. Bratko. Computer analysis of world chess champions. *ICGA Journal*, 29(2):3–14, 2006.
- [GB07] M. Guid and I. Bratko. Factors affecting diminishing returns for searching deeper. *ICGA Journal*, 30(2):65–73, 2007.
- [GGM93] D. Gadwal, J.E. Greer, and G.I. McCalla. Tutoring bishop-pawn endgames: An experiment in using knowledge-based chess as a domain for intelligent tutoring. *Applied Intelligence*, 3(3):207–224, 1993.
- [GMB03] D. Gomboc, T.A. Marsland, and M. Buro. Evaluation function tuning via ordinal correlation. In *ACG*, pages 1–18, 2003.
- [GMK⁺08] M. Guid, M. Možina, J. Krivec, A. Sadikov, and I. Bratko. Learning positional features for annotating chess games: A case study. In *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2008.
- [GMSB10] M. Guid, M. Možina, A. Sadikov, and I. Bratko. Deriving concepts and strategies from chess tablebases. In *ACG 2009*, volume 6048 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2010.
- [GPB08] M. Guid, A. Perez, and I. Bratko. How trustworthy is CRAFTY's analysis of world chess champions? *ICGA Journal*, 31(3):131–144, 2008.
- [HACN90] F. Hsu, T. Anatharaman, M. Campbell, and A. Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, 1990.
- [Haw07] G. Haworth. Gentlemen, stop your engines! *ICGA Journal*, 30(3):150–156, 2007.

- [HB96] H. Herbeck and W. Barth. An explanation tool for chess endgames based on rules. *ICCA Journal*, 19(2):75–82, 1996.
- [Hei98] E.A. Heinz. DarkThought goes deep. *ICCA Journal*, 21(4):228–244, 1998.
- [Hei99a] E.A. Heinz. Modeling the “go deep” behaviour of Crafty and DarkThought. In H.J. Van den Herik and B. Monien, editors, *Advances in Computer Chess 9*, pages 59–71. Universiteit Maastricht, 1999.
- [Hei99b] E.A. Heinz. Self-play in computer chess revisited. In H.J. Van den Herik and B. Monien, editors, *Advances in Computer Chess 9*, pages 73–91. Universiteit Maastricht, 1999.
- [Hei01] E.A. Heinz. Self-play, deep search and diminishing returns. *ICGA Journal*, 24(2):75–79, 2001.
- [Hei03] E.A. Heinz. Follow-up on self-play, deep search, and diminishing returns. *ICGA Journal*, 26(2):75–80, 2003.
- [HN97] R. Hyatt and M.M. Newborn. Crafty goes deep. *ICGA Journal*, 20(2):79–86, 1997.
- [HRdF10] G. Haworth, K. Regan, and G. di Fatta. Performance and prediction: Bayesian modelling of fallible choice in chess. In *ACG 2009*, volume 6048 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2010.
- [HvBK08] R. Higdon, G. van Belle, and E. Kolker. A note on the false discovery rate and inconsistent comparisons between experiments. *Bioinformatics*, 24(10):1225–1228, 2008.
- [JS99] A. Junghanns and J. Schaeffer. Search versus knowledge in game-playing programs revisited. In *15th International Joint Conference on Artificial Intelligence, Proceedings*, volume 1, pages 692–697. Morgan Kaufmann, 1999.
- [JSB⁺97] A. Junghanns, J. Schaeffer, M. Brockington, Y. Björnsson, and T.A. Marsland. Diminishing returns for additional search in chess. In H.J. Van den Herik and B. Monien, editors, *Advances in Computer Chess 9*, pages 53–67. Universiteit Maastricht, 1997.
- [Kar08] T. Karlsson. The Swedish rating list. *ICGA Journal*, 31(4):255, 2008.
- [Kas06] G. Kasparov. *My Great Predecessors, Parts 1-5*. Everyman Chess, London, 2003-2006.

- [KBW83] J. Kulik, R.L. Bangert, and G. Williams. Effects of computer-based teaching on secondary school students. *Journal of Educational Psychology*, 75(1):19–26, 1983.
- [Koh95] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *14th International Joint Conferences on Artificial Intelligence (IJCAI 1995), Proceedings*, pages 1137–1143. Morgan Kaufmann, 1995.
- [KS06] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293. Springer, 2006.
- [KWM97] D.E. Kieras, S.D. Wood, and D.E. Meyer. Predictive engineering models based on the epic architecture for a multimodal high-performance human-computer interaction task. *ACM Trans. Comput.-Hum. Interact.*, 4(3):230–275, 1997.
- [LC06] P. Langley and D. Choi. A unified cognitive architecture for physical agents. In *AAAI’06: proceedings of the 21st national conference on Artificial intelligence*, pages 1469–1474, 2006.
- [LGB05] M. Lustrek, M. Gams, and I. Bratko. Why minimax works: an alternative explanation. In *19th International Joint Conference on Artificial Intelligence, Proceedings*, volume 1, pages 212–217. Professional Book Center, 2005.
- [LLR09] P. Langley, J. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.
- [LM96] D. Levy and T. Marsland. The ICCA best annotation award for 1995. *ICCA Journal*, 19(2):135–136, 1996.
- [LNR87] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
- [LS95] P. Langley and H.A. Simon. Applications of machine learning and rule induction. *Commun. ACM*, 38(11):54–64, 1995.
- [MGK⁺08] M. Možina, M. Guid, J. Krivec, A. Sadikov, and I. Bratko. Fighting knowledge acquisition bottleneck with argument based machine learning. In *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 234–238. IOS Press, 2008.

- [MGSB09] M. Možina, M. Guid, A. Sadikov, and I. Bratko. Goal-Based Rule Learning. Technical report, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, 2009. Also available as <http://www.ailab.si/matej/KBNK/GBRL.pdf>.
- [Mur99] T. Murray. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10:98–129, 1999.
- [MvB07] M. Možina, J. Žabkar, and I. Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10/15):922–937, 2007.
- [New75] M.M. Newborn. *Computer Chess*. Academic Press, New York, 1975.
- [New85] M.M. Newborn. A hypothesis concerning the strength of chess programs. *ICCA Journal*, 8(4):209–215, 1985.
- [New90] A. Newell. *Unified theories of cognition*. Harvard University Press, Cambridge, MA, USA, 1990.
- [Nim06] A. Nimzovich. *The Blockade*. Hardinge Simpole Limited, 2006.
- [NS72] A. Newell and H.A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [Nun95] J. Nunn. *Secrets of Minor-Piece Endings*. Batsford, 1995.
- [Nun02] J. Nunn. *Secrets of Pawnless Endings*. Gambit Publications Limited, 2002.
- [Pea83] J. Pearl. On the nature of pathology in game searching. *Artificial Intelligence*, 20(4):427–453, 1983.
- [Pea84] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.
- [PV02] H. Prakken and G. Vreeswijk. *Handbook of Philosophical Logic, second edition*, volume 4, chapter Logics for Defeasible Argumentation, pages 218–319. Kluwer Academic Publishers, Dordrecht etc, 2002.
- [Ros05] S.M. Ross. *Introductory Statistics*. Elsevier Academic Press, 2005.
- [Roy88] A.J. Roycroft. Expert against oracle. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11*, pages 347–373. Oxford University Press, Oxford, UK, 1988.

- [Sad05] A. Sadikov. *Propagation of Heuristic Evaluation Errors in Game Graphs*. PhD thesis, University of Ljubljana, Faculty of Computer and Information Science, 2005.
- [SB06] A. Sadikov and I. Bratko. Search versus knowledge revisited again. In *Proceedings of the 5th International Conference on Computers and Games (CG2006)*, 2006.
- [Sch86] J. Schaeffer. *Experiments in search and knowledge*. PhD thesis, University of Waterloo, Canada, 1986.
- [Sei94] D. Seidel. Self-annotating elementary endgames. *ICCA Journal*, 17(2):51–62, 1994.
- [SK98] A. Scheucher and H. Kaindl. Benefits of using multivalued functions for minimaxing. *Artificial Intelligence*, 99(2):187–208, 1998.
- [SMG⁺07] A. Sadikov, M. Možina, M. Guid, J. Krivec, and I. Bratko. Automated chess tutor. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2007.
- [Son] J. Sonas. Chessmetrics. <http://www.chessmetrics.com>.
- [SS98] R. Sison and M. Shimura. Student modeling and machine learning. *International Journal of Artificial Intelligence in Education*, 9:128–158, 1998.
- [Ste05] J.R. Steenhuisen. New results in deep-search behaviour. *ICGA Journal*, 28(4):203–213, 2005.
- [Tho82] K. Thompson. Computer chess strength. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, pages 55–56. Pergamon Press, 1982.
- [Tho86] K. Thompson. Retrograde analysis of certain endgames. *International Computer Chess Association Journal*, 9(3):131–139, 1986.
- [Twi92] M. Twidale. Knowledge acquisition for intelligent tutoring systems. In T. Bösser F.L. Engel, D.G. Bouwhuis and G. d’Ydewalle, editors, *Cognitive modelling and interactive environments in language learning*. Springer-Verlag, 1992.
- [vdH83a] H.J. van den Herik. *Computerschaak, Schaakwereld en Kunstmatige Intelligentie*. PhD thesis, Delft University of Technology – Universidade Nova de Lisboa (ITQB/UNL), Academic Service, The Hague, 1983.

- [vdH83b] H.J. van den Herik. Representation of experts' knowledge in a subdomain of chess intelligence. In *IJCAI*, pages 252–255, 1983.
- [vdHH86] H.J. van den Herik and I.S. Herschberg. Omniscience, the rulegiver? In M. Somalvico B. Pernici, editor, *Proceedings of L'Intelligenza Artificiale Ed Il Gioco Degli Scacchi, III Convegno Internazionale*, pages 1–17, 1986.
- [vdHUvR02] H.J. van den Herik, J.W.H.M. Uiterwijk, and J. van Rijswijk. Games solved: Now and in the future. *Artificial Intelligence*, 134:277–311, 2002.
- [VLS⁺05] K. VanLehn, C. Lynch, K.S. Schulze, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill. The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3):147–204, 2005.
- [Wat99] J. Watson. *Secrets of Modern Chess Strategy*. Gambit Publications, 1999.
- [WBS08] M.H. Winands, Y. Björnsson, and J. Saito. Monte-carlo tree search solver. In *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*. Springer, 2008.
- [Woo08] B.P. Woolf. *Building Intelligent Interactive Tutors*. Morgan Kauffman, New York, 2008.
- [Yaz86] M. Yazdani. Intelligent tutoring systems survey. *Artificial Intelligence Review*, 1(1):43–52, 1986.