University of Bath

PHD

Stochastic optimisation methods for cost-effective quality assessment in health

Fouskakis, Dimitris

Award date:
2001

Awarding institution:
University of Bath

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. May. 2019

Stochastic Optimisation Methods for Cost-Effective Quality Assessment in Health

submitted by

Dimitris Fouskakis

for the degree of Ph.D.

of the

University of Bath

2001

COPYRIGHT

Attention is drawn to the fact that copyright of this thesis rests with its author. This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signature of Author ....

Dimitris Fouskakis

UMI Number: U131765

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U131765
Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

UNIVERSITY OF BATH
LIBRARY
35 | 20 JUN 2001
PHD

To

My Mother and Sister

Abstract

Patient sickness at admission to hospital is traditionally measured by using logistic regression of mortality within 30 days of admission on $O(100)$ sickness indicators to construct a sickness scale, employing classical variable selection methods to find an "optimal" subset of 10–20 indicators. Such methods ignore the considerable differences among the sickness indicators in cost of data collection, which become crucial when admission sickness is used to drive programmes (now under consideration in several countries, including the UK and US) that attempt to identify substandard hospitals by comparing observed and expected mortality (given admission sickness). When both data-collection cost and accuracy of prediction of 30-day mortality are considered, a large optimisation problem arises in which costly variables that do not predict well enough should be omitted from the scale.

In this dissertation I take a Bayesian decision-theoretic approach (based on maximisation of expected utility) to solving this optimisation problem, using data from a large US study of quality of hospital care in the 1980s. I use *genetic algorithms* (GA), *simulated annealing* (SA), *tabu search* (TS), and other methods from the optimisation literature to find (near-)optimal subsets of predictor variables. I find that (i) the best versions of GA outperform the best versions of TS, with the advantage for GA growing as the number p of variables available for constructing the sickness scale increases; (ii) both GA and TS are sharply better than SA in this problem for all values of p studied; and (iii) optimal subsets of variables that compromise between data collection costs and predictive accuracy have the potential to generate large cost savings in quality assessment programmes.

This work (a) provides a new perspective on variable selection in generalised linear models, (b) offers new insights into the comparative advantages and flaws of competing optimisation methods, and (c) produces results of direct use in health policy.

Summary

An expert is a person who has made all the mistakes which can be made in a very narrow field.

— Niels Bohr

Introduction. An important topic in health policy is the assessment of the quality of health care offered to hospitalised patients. Quality of care is usually thought to depend mainly on three ingredients: (i) *process*, which is what health care providers do on behalf of patients, (ii) *outcomes*, which are what happens to patients as a result of the care they receive, and (iii) patient *sickness at admission*, because the appropriateness of outcomes cannot be judged without taking account of the burden of illness brought to the hospital by its patients.

A direct audit of the processes of care is usually regarded as the single most informative component in an evaluation of quality, but process is much more expensive to measure than outcomes or admission sickness. Interest has therefore focused in recent years, in countries such as the United States and the United Kingdom, on an indirect method of assessment—which might be termed the *input-output* approach[1]—in which hospital outcomes (for instance, *death within 30 days of admission*) are compared after adjusting for differences in inputs (sickness at admission). The idea is to treat what goes on inside the hospital—process—as a black box, with the contents of the box inferred by examining its outputs after taking account of its inputs.

Indirect measurement of quality of health care. In practice, to indirectly measure quality of care at any given moment in time, this strategy proceeds by (a) taking a sample of hospitals and a sample of patients in the chosen hospitals, (b) obtaining death outcomes for the sampled patients (for example, from central government data bases), (c) extracting information on admission sickness from the medical records of these patients, (d) forming an expected mortality rate for each

[1] In the UK this approach is also referred to as *league-table quality assessment*, by analogy with the process of ranking football teams.

hospital based on (c), and (e) comparing observed and expected mortality to identify unusual hospitals (on both the "good" and "bad" ends of the spectrum). Since this would involve abstracting data from the charts of many thousands of patients if it were attempted on a large scale, the *cost-effective* measurement of admission sickness is crucial to this approach.

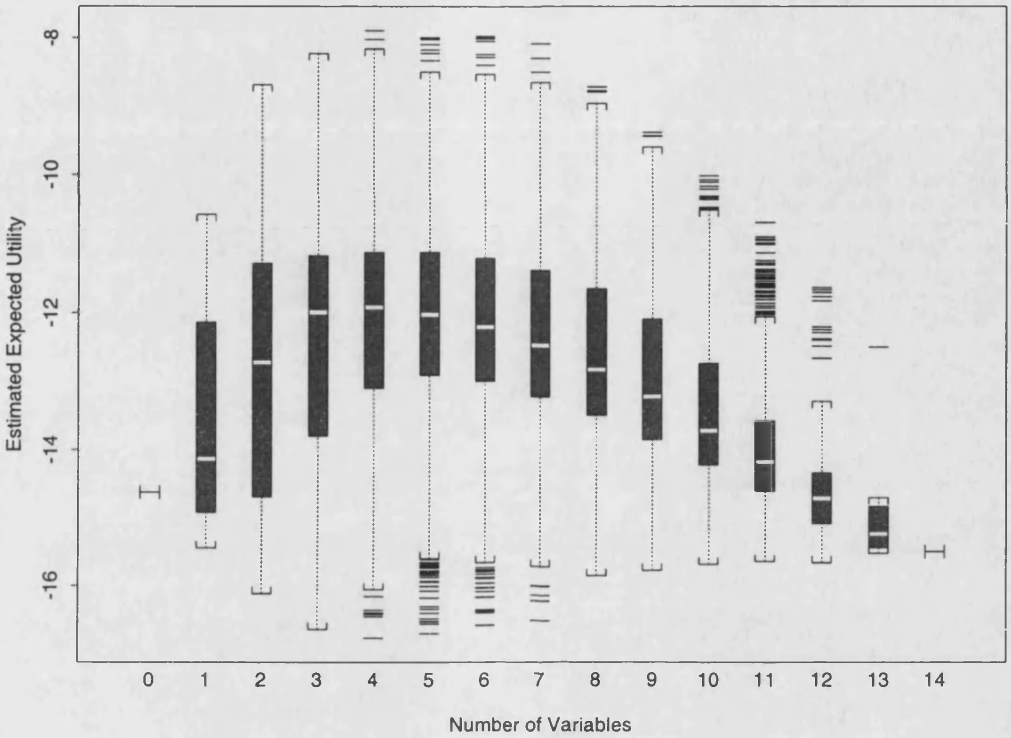
Quality of care assessment is a highly disease-specific activity: for instance, the right admission sickness variables to examine for pneumonia would be quite different from those for heart attack. With any given disease there will be on the order of 100 separate variables potentially available in the medical record that are directly or indirectly related to admission sickness. In the case of pneumonia, for example, on which I focus exclusively in this dissertation, a list of the important variables from a clinical perspective would include such things as the systolic blood pressure on day 1 of admission, the presence or absence of prior respiratory failure, and the blood urea nitrogen level (a measure of kidney functioning).

Previous approaches to constructing admission sickness scales. The standard method for creating an expected mortality rate from these admission sickness inputs in any given nation, such as the US or UK, is logistic regression, with 30–day death as the outcome, and using a nationally-representative sample of patients to normalise the expectation to average care across the nation. Typically a frequentist variable-selection method—such as all-subsets regression—is employed to find a parsimonious and clinically reasonable subset of the available sickness variables. In a major US study conducted by the Rand Corporation, of quality of hospital care for elderly patients in the late 1980s, this approach was used to reduce the list of 83 available sickness indicators for pneumonia down to a core of 14 predictors.

As good as the resulting scale may be on grounds of simplicity and ease of clinical communication, I take the view in this dissertation that—when the goal is the creation of a sickness scale that may be used prospectively to measure quality of care on a new set of patients not yet examined—the original Rand approach is sub-optimal, because it takes no account of differences in the *cost of data collection* among the available predictors (which varied for pneumonia from *10 seconds* to *15 minutes* of abstraction time per variable). The Rand approach represents a kind of benefit-only analysis; I propose a cost-benefit analysis, in which variables are chosen for the final scale only when they predict mortality well enough given how much they cost to collect.

A large optimisation problem. Weighing data-collection costs against the accuracy of prediction creates a large optimisation problem which cannot be solved

Figure 0-1: *Boxplots of estimated expected utility as a function of number of predictors, based on the Rand scale with* $p = 14$ *variables.*



by brute-force enumeration: for example, when $p = 83$ it is necessary to compare $2^p \doteq 9.7 \cdot 10^{24}$ subsets of sickness variables, and even at the rate of 100 subsets examined per second—which is far faster than present computational resources permit with desktop workstations—it would take more than $3 \cdot 10^{15}$ years to find the optimal subset by looking at all of them.

Suppose (a) the 30–day mortality outcome y_i and data on p sickness indicators $(x_{i1}, \ldots, x_{ip})$ have been collected on n individuals sampled randomly from a population $\mathcal{P}$ of patients with a given disease, and (b) the goal is to predict the death outcome for n^* new patients who will in the future be sampled randomly from $\mathcal{P}$, (c) on the basis of some or all of the predictors x_j, when (d) the marginal costs of data collection per patient $c_1, \ldots, c_p$ for the x_j vary considerably. What is the best subset of the x_j to choose, if a fixed amount of money is available for this task and you are rewarded based on the quality of your predictions?

A Bayesian solution. To solve this problem I use a Bayesian decision-theoretic approach based on maximisation of expected utility. The utility function I use has two components, one to quantify data collection costs and one to keep track of predictive successes and failures on future patients. The data on which I demonstrate this method in the dissertation consist of a representative sample of $n = 2,532$ elderly American patients hospitalised in the period 1980–86 with pneumonia, taken from the Rand study described above. Since data on future patients are not available, I use a cross-validation approach in which (i) a random subset of m observations is drawn for creation of the mortality predictions and (ii) the quality of those predictions is assessed on the remaining $(n - m)$ observations (with this approach the expectation in the calculation of expected utility is over all possible cross-validation splits).

I have performed a brute-force evaluation of all $2^{14} = 16,384$ possible subsets of the predictors in the Rand scale based on averages across 500 random splits in each case, with results as in Figure 0–1. It is clear that the full Rand scale is sharply suboptimal, with scales based on 4–7 variables saving almost US$8 per patient, which would translate into many millions of dollars or pounds if the league-table approach were to receive widespread implementation in the US or UK.

Stochastic optimisation methods. Of course, this kind of brute-force enumeration will not work with the full set of $p = 83$ variables available. In the main part of the dissertation I compare a variety of stochastic optimisation methods—including *simulated annealing* (SA), *genetic algorithms* (GA), and *tabu search* (TS)—both in the $p = 14$ case where we know the right answer and in the full $p = 83$ case where we do not. I have examined the geometry of the solution space; studied the optimal allocation of CPU resources between (i) searching for new models and (ii) increasing the number of cross-validation splits to obtain a better estimate of the quality of models already visited; conducted a variety of sensitivity analyses to examine the stability of my findings across alternative formulations; and performed extensive simulations to obtain recommendations on the input settings to the optimisation methods which maximise their performance. Results are as follows.

- The most up-to-date variations of GA—featuring (i) *elitist strategies*, (ii) *uniform* or *highly uniform crossover operators*, (iii) the retention of 100% of the previous population in each repetition, and (iv) the use of small to moderate population sizes (e.g., 30–50)—outperform TS by an amount which is small with $p = 14$ but large with $p = 83$. GA was the only method in the 83–variable case able to find good models in a modest amount of CPU time.

- However, with $p = 14$ GA is also the method whose performance depends most critically on intelligent choice of input settings. TS is far more robust than GA in the 14–variable case to sub-optimal choice of user-defined inputs.

- Both GA and TS dramatically dominate SA for both values of p examined.

These findings are interesting because TS is almost unknown to statisticians and there is a substantial body of folklore in statistics expressing the view that GA is inferior (even to SA) in many optimisation problems.

The work presented here (a) provides a new perspective on variable selection in generalised linear models, (b) offers new insights into the comparative advantages and flaws of competing optimisation methods, and (c) produces results of direct use in health policy.

Acknowledgements

> The mind is like a parachute: it functions only when it's open.
>
> — Anonymous

I feel deeply indebted to my supervisor and dear friend, Professor David Draper, whose research interests served as a springboard for this thesis. I would also like to thank him for his tireless assistance and substantial support throughout both my MSc and PhD. Thanks also go to Dr Katherine Kahn and other physicians at the Rand Corporation for providing data and clinical expert judgement as inputs to this project, and to Dr William Browne for his comments on the dissertation. Special thanks are due to the University of Bath for their financial support.

I would also like to thank my fellow office mates, past and present, especially David Esparza and Fas Yousaf, for their friendship and humour, and all my friends, both in Greece and the UK, for their encouragement and emotional support.

Furthermore, my special thanks go to my great friend and housemate Panos, for simply being the best friend I had during all these years in Bath, and of course to Katerina and Lampros for making this difficult year very special. Last but not least, I would like to thank my girlfriend Vhyna for her devoted love, support and understanding, and for making me so happy through the last two years.

To conclude with, I dedicate the present thesis to my mother Vaso and my sister Kelly, for supporting me greatly during all these years with their love and affection.

Contents

Dedication i

Abstract ii

Summary iii

Acknowledgements viii

1 Introduction 1

1.1 The health policy background 1

1.2 Variable selection 3

1.3 Contents 5

2 Problem formulation 8

2.1 Decision-theoretic approach 8

2.1.1 Data-collection utility 9

2.1.2 Predictive utility 9

2.2 The goals of this project 12

3 Stochastic optimisation 14

3.1 Introduction 14

3.2 Simulated annealing (SA) 16

3.2.1 Generic and problem-specific decisions 18

3.2.2 Modifications 21

3.2.3 Messy simulated annealing (MSA) 27

3.2.4 SA Summary 30

3.3 Threshold acceptance (TA) 31

3.4 Genetic algorithms (GA) 33

3.4.1 Biological terminology 33

3.4.2 The algorithm 34

3.4.3 The Schema Theorem 36
3.4.4 Implementation of GA 38
3.4.5 Modifications 40
3.4.6 Genitor algorithm 52
3.4.7 CHC adaptive search algorithm 52
3.4.8 GA summary 53
3.5 Tabu search (TS) 54
3.5.1 The algorithm 55
3.5.2 Implementation and modifications of TS 59
3.5.3 TS summary 65
3.6 Summary of all optimisation methods studied 65

4 Results in the case $p = 14$ 67

4.1 Introduction 67
4.2 Full enumeration results 68
4.3 Geometry of the solution space 70
4.4 Optimal choice of N 73
4.5 Comparison of optimisation methods: preliminary results 75
4.6 A simulation experiment with $p = 14$ 78
4.6.1 Tabu search 79
4.6.2 Simulated annealing 80
4.6.3 Genetic algorithm 82
4.7 Comparison of optimisation methods: final results for $p = 14$ 90
4.8 Sensitivity analyses 91
4.8.1 Second full-enumeration run: different random number seed . 91
4.8.2 Third full-enumeration run: different choice of $\left(\frac{n_M}{n}, \frac{n_V}{n}\right)$ . . . 92
4.8.3 Quantitative comparison of the full-enumeration runs 93
4.8.4 Penalties and rewards for prediction accuracy and marginal costs per variable 94
4.8.5 Interaction terms 95

5 Results in the case $p = 83$ 97

5.1 Introduction 97
5.2 One-week results 101
5.3 A simulation experiment with $p = 83$ 104
5.3.1 Tabu search 106
5.3.2 Simulated annealing 106

5.3.3 Genetic algorithm 109
5.4 Results with 24 hours of CPU time 111
5.4.1 Tabu search 111
5.4.2 Simulated annealing 112
5.4.3 Genetic algorithm 113
5.5 Comparison of optimisation methods: final results with $p = 83$ 113

6 Conclusions and extensions 116
6.1 Summary of the project and its main findings 116
6.2 Suggestions for future work 119
6.2.1 Improved simulated annealing 123

Bibliography 132

List of Figures

0-1 *Boxplots of estimated expected utility as a function of number of predictors, based on the Rand scale with* $p = 14$ *variables.* v

3-1 *The four temperature schedules in Table 3.1, with* $T_0 = 1.0, T_f = 0.1$, *and* $M = 1,000$. 19

3-2 *An example of decoding of the M-chromosome* $c = [(3,0),(1,1),(3,1),(2,0),(5,1)]$ *to the chromosome* $(1,0,0,1,1)$ *with respect to the template* $(1, 0,0,1,1)$. 28

3-3 *Example of mutation of the M-chromosome* $c = [(3,0),(1,1),(3,1),(2,0),(5,1)]$. *Vertical arrows indicate places where allelic and genic mutations are applied. Allelic mutation switches the bit, while genic mutation randomly changes the position of the bit.* 29

4-1 *Estimated expected utility as a function of number of predictors retained, from the first full-enumeration run with* $p = 14$. 70

4-2 *Tree of adjacent models* ($k = 4$) *expanded out to four levels, with the neighbourhood structure induced by moves based on one-bit flips. The horizontal and vertical scales are arbitrary.* 71

4-3 *Perspective plot of the expected utility "surface" for the 4–variable tree expanded out to four levels.* 72

4-4 *Performance of SA on a run that found the global optimum in the* $p = 14$ *case, allowing the method 24 hours of CPU time at 400 MHz.* 73

4-5 *Actual expected utility as a function of N for a random-walk search strategy (the horizontal scale is logarithmic).* 74

4-6 *Parallel boxplots comparing the three optimisation methods in the 14–variable case.* 92

4-7 *Like Figure 4–1 but with a different random number seed.* 93

4-8 *Like Figures 4–1 and 4–7 but with* $\left(\frac{n_M}{n}, \frac{n_V}{n}\right) = \left(\frac{1}{3}, \frac{2}{3}\right)$ *instead of* $\left(\frac{2}{3}, \frac{1}{3}\right)$. 93

5-1 *Estimated (real) expected utility as a function of number of predictors retained, based on the 3,000 best models found from the one-week runs with* $p = 83$. 103

6-1 *Parallel boxplots comparing GA, ISA, and TS in the 14–variable case.* 129

List of Tables

1.1 *The full list of the 83 variables relevant to sickness at admission for pneumonia patients, using Rand naming conventions.* 4

1.2 *The final variables for pneumonia chosen by Rand.* 6

2.1 *Cross-tabulation of actual versus predicted death status. The left-hand table records the monetary rewards and penalties for correct and incorrect predictions; the right-hand table summarises the frequencies in the* 2×2 *tabulation.* 11

3.1 *Families of temperature schedules for simulated annealing.* 19

3.2 *Illustrative rules for tabu list size.* 62

4.1 *The 14 variables in the Rand pneumonia admission sickness scale, together with their approximate data collection costs per patient and correlation with 30–day death (CHF = congestive heart failure).* . . . 69

4.2 *Renaming the 16 models with* $p = 4$. 72

4.3 *Preliminary comparison of GA, MSA, SA, TS, and TA. The adaptive-*N^* *method was used in all cases. Boldface indicates the best result in each column for each CPU time constraint.* 77

4.4 *Results of the simulation study for TS with* $p = 14$ *(part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of* p_{20}. 81

4.5 *Results of the simulation study for TS with* $p = 14$ *(part 2).* 82

4.6 *Results of the simulation study for SA with* $p = 14$ *(part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of* p_{20}. 83

4.7 *Results of the simulation study for SA with* $p = 14$ *(part 2).* 84

4.8 *Results of the simulation study for SA with* $p = 14$ *(part 3).* 85

4.9 *Results of the simulation study for GA with* $p = 14$ *(part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of* p_{20}. 87

4.10 *Results of the simulation study for GA with* $p = 14$ *(part 2).* 88

4.11 *Results of the simulation study for GA with* $p = 14$ *(part 3).* 89

4.12 *Results of the simulation study for GA with* $p = 14$ *(part 4).* 90

4.13 *Comparison of the 20 best models of the first full-enumeration run with the other 2 runs.* 94

4.14 *Pairwise rank correlations across the three full-enumeration runs as a function of the number of variables in the model.* 95

5.1 *The full set of 83 variables, together with their approximate data collection costs per patient, correlation r with 30–day death, and presence in the original Rand 14–variable scale (part 1).* 98

5.2 *The full set of 83 variables (part 2).* 99

5.3 *Input settings for the one-week runs.* 102

5.4 *Distribution of model dimension in the 3,000 best models from the one-week runs, by optimisation method.* 102

5.5 *Summary of the 3,000 best and the 100 best models found in the one-week runs.* 104

5.6 *Input settings and results for the 3–hour runs of TS in the simulation experiment with* $p = 83$ *(SDs in parenthesis).* 107

5.7 *Input settings and results for the 3–hour runs of SA in the simulation experiment with* $p = 83$ *(SDs in parenthesis).* 108

5.8 *Input settings and results for the 3–hour runs of GA in the simulation experiment with* $p = 83$ *(SDs in parenthesis; for explanation of other symbols see Table 5.6).* 110

5.9 *Input settings and results for the 24–hour run of TS in the simulation experiment with* $p = 83$. 111

5.10 *Input settings and results for the 24–hour run of SA in the simulation experiment with* $p = 83$. 112

5.11 *Input settings and results for the 24–hour run of GA in the simulation experiment with* $p = 83$. 113

5.12 *Comparison of the total number of utility evaluations achieved by the three optimisation methods in the 24–hour runs.* 114

6.1 *An ad hoc measure of the desirability of a variable in compromising between predictive accuracy and data collection costs, in the case* $p = 14$.122

6.3 *Variable desirability values d_j and the corresponding p_j^{in} values in the 14–variable case, using the desirability-to-probability transformation given by equation (6.1).* 124

6.4 *Results of the simulation study for improved SA with $p = 14$ (part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of p_{20}.* 126

6.5 *Results of the simulation study for improved SA with $p = 14$ (part2).* . . 127

6.6 *Results of the simulation study for improved SA with $p = 14$ (part3).* . . 128

6.7 *Input settings and results for the 3–hour runs of ISA in the simulation experiment with $p = 83$ (SDs in parenthesis).* 130

Chapter 1

Introduction

The last thing one knows when writing a book is what to put first.

— Blaise Pascal

1.1 The health policy background

In 1983 the US federal government established a *Prospective Payment System* (PPS) in order to control the costs of reimbursing hospitals for care of elderly and handicapped patients under the Medicare programme. Prior to 1983, under the former *retrospective* payment system, the care was provided first and then the government reimbursed the hospital. To the extent that this system contained counterproductive economic incentives, the tendency was to *over*-treat patients in an attempt to increase profits. Under PPS, a fixed price for each episode of care was established, so that in effect hospitals knew before the care was provided how much they would be paid. At the time PPS began, concern arose that the new tendency would be to *under*-treat patients, giving rise to the possibility of a decline in the quality of hospital care offered under Medicare.

The assessment of the quality of health care offered to hospitalised patients is an important topic in health policy. Quality of care is usually thought to depend mainly on three ingredients (Donabedian 1981): *process*, which is what health care providers do on behalf of patients, *outcomes*, which are what happens to patients as a result of the care they receive, and patient *sickness at admission*, because the appropriateness of outcomes cannot be judged without taking account of the burden of illness brought to the hospital by the patient.

In 1985, the Rand Corporation began a study (Kahn et al. 1990b) of the effects of PPS on quality of care. Approximately 16,500 elderly patients, aged 65 and over, hospitalised with one or another of {congestive heart failure, acute myocardial infarction, hip fracture, pneumonia, cerebrovascular accident, depression}, were selected in a nationally representative manner from five different states, each from a different geographic region of the nation: California, Florida, Indiana, Pennsylvania and Texas. Full details on the sampling plan are available in (Draper et al. 1990).

A direct audit of the processes of care is usually regarded as the single most informative component in an evaluation of quality, but process is much more expensive to measure than outcomes or admission sickness (Kahn et al. 1990a). Interest has therefore focused in recent years, in countries such as the United States and the United Kingdom, on an indirect method of assessment—which has been termed the *input-output* approach (Draper 1995)—in which hospital outcomes (for instance, *death within 30 days of admission*) are compared after adjusting for differences in inputs (sickness at admission). (In the UK this approach is also referred to as *league-table quality assessment* (Goldstein and Spiegelhalter 1996), by analogy with the process of ranking football teams.) The idea is to treat what goes on inside the hospital—process—as a black box, with the contents of the box inferred by examining its outputs after taking account of its inputs (Daley et al. 1988).

In practice, to indirectly measure quality of care at any given moment in time, this strategy proceeds by (a) taking a sample of hospitals and a sample of patients in the chosen hospitals, (b) obtaining death outcomes for the sampled patients (for example, from central government data bases), (c) extracting information on admission sickness from the medical records of these patients, (d) forming an expected mortality rate for each hospital based on (c), and (e) comparing observed and expected mortality to identify unusual hospitals (in both tails of the distribution). Since this would involve abstracting data from the charts of many thousands of patients if it were attempted on a large scale, the *cost-effective* measurement of admission sickness is crucial to this approach.

Quality of care assessment is a highly disease-specific activity: for instance, the best admission sickness variables to examine for pneumonia would be quite different from those for heart attack. With any given disease there will be on the order of 100 separate variables potentially available in the medical record that are directly or indirectly related to admission sickness. In the case of pneumonia, for example, on which I focus exclusively in this dissertation, a list of the important variables from a clinical perspective (Kahn et al. 1990b) would include such things as systolic blood pressure on day 1 of admission, the presence or absence of prior respiratory failure, and the blood urea nitrogen level (a measure of kidney functioning).

The standard method for creating an expected mortality rate from these admission sickness inputs is logistic regression (Hosmer and Lemeshow 1989), with 30–day death as the outcome, and using a nationally-representative sample of patients to normalise the expectation to average care across the nation. Typically a frequentist variable-selection method—such as all-subsets regression (Weisberg 1985)—is employed to find a parsimonious and clinically reasonable subset of the available sickness variables. In a major US study conducted by the Rand Corporation, of quality of hospital care for elderly patients in the late 1980s, this approach was used (Keeler 1993) to reduce the list of 83 available sickness indicators for pneumonia down to a core of 14 predictors.

As good as the resulting scale may be on grounds of simplicity and ease of clinical communication, I take the view in this dissertation that—when the goal is the creation of a sickness scale that may be used prospectively to measure quality of care on a new set of patients not yet examined—the original Rand approach is sub-optimal, because it takes no account of differences in the *cost of data collection* among the available predictors (in terms of abstraction time per variable, which can readily be converted into costs, the range for pneumonia across the sickness indicators was from 10 seconds to 15 minutes). The Rand approach represents a kind of benefit-only analysis; I propose here a cost-benefit analysis, in which variables are chosen for the final scale only when they predict mortality well enough given how much they cost to collect.

In this dissertation I have chosen one of the Rand diseases, pneumonia, to implement a method proposed by (Draper 1996) that uses logistic regression and Bayesian utility analysis to construct a scale measuring sickness at admission that balances accuracy and cost. Scales constructed with this method would help in the process of league table quality assessment, by making the best use of public money to identify substandard hospitals.

1.2 Variable selection

The Rand study (Kahn et al. 1990b) used disease-specific abstraction forms to collect data about sickness at admission from the medical records of roughly 2,750 hospitalised patients per disease, although mortality information was only available on about 2,550 patients per disease. Table 1.1 provides a full list of the 83 variables relevant to sickness at admission gathered for pneumonia patients.

Rand used literature review, clinical judgement and disease-specific consensus panels to identify variables that have been considered important clinical predictors of either in-hospital death or death within 30 days of hospitalisation. Variables were

Table 1.1: *The full list of the 83 variables relevant to sickness at admission for pneumonia patients, using Rand naming conventions.*

Name	Meaning	Name	Meaning
p_sbpd1	systolic blood pressure score	sage2	age of patient
sbun	blood urea nitrogen	pc_comas	APACHE II coma score
p_sobd1	shortness of breath day 1	pc_albms	Serum albumin score
pc_resps	respiratory distress	septic	septic complications
pc_prfs	prior respiratory failure	pc_phsps	recently hospitalised
pc_blps	racbilateral process score	ctemp	initial temperature
p_hrd1	heart rate day 1	p_cpd1	chest pain day 1
pc_cards	cardiomegaly score	pc_effs	plural effusion score
pc_xpnes	pneumonia CXR score	pc_ambls	ambulatory score
pc_endos	endocarditis at admission	pc_cpks	CPK score
pc_pilds	prior interstitial lung disease	pc_antis	prior antibiotics score
pc_hmeos	home oxygen use	pc_pnems	prior pneumonectomy
pc_ptrcs	prior tracheostomy	pc_amins	prior aminophylline score
pc_hems	hematologic history score	pc_pcncs	cancer score
pc_hrs	APACHE heart rate score	pc_crods	Corodaker score
pc_dthxs	disease of thorax	pc_myels	multiple myeloma
pc_immns	immunocompromised	pc_resds	residence score
pc_hephs	hepatobiliary history	pc_renls	renal history score
pc_rrs	APACHE respiratory rate score	dpc_nlcs	new lung score
pc_asprs	co-morbid aspiration score	pc_nas	APACHE sodium score
pc_ahcts	APACHE hematocrit score	pc_wbcs	APACHE WBC score
pc_oxys	APACHE oxygenation score	pc_pcvas	CVA score
pc_ks	APACHE potassium score	sbp	systolic BP (admission)
pc_xchfs	CHF chest X-ray score	pc_aps2	Total APACHE II score
p_rrd1	respiratory rate day 1	p_dbpd1	DIA blood press day 1
p_cond1	confusion day 1	pc_pulms	pulm. vasc. cong. score
pc_co3s	APACHE venus bicarb score	pc_pedms	pulmonary edema score
pc_chfsm	sum of CHF components	pc_flus	influenza score
pc_ers	arrest in ER score	pc_bilis	biliribin score
pc_pbcs	positive blood culture	pc_pucs	positive urine culture
pc_weezs	wheezing at admission	pc_body	body system count
pc_pcpds	morbid prior COPD score	pc_pphps	morbid pulm. hosp. score
pc_pcrhs	co-morbid cirrh score	pc_pchfs	co-morbid CHF score
pc_arrys	co-morbid arrhythmias score	pc_smkrs	co-morbid smokers score
pc_alchs	co-morbid alcoholism score	pc_phs	APACHE PH score
pc_ngts	co-morbid NGTS score	pc_sters	co-morbid steroids score
pc_rctot	sum of morbid+comorbid	pc_crdhs	cardiac history score
pc_neurs	neurologic history score	pc_oncos	oncologic history score
pc_imuns	immunologic history score	pc_muscs	musculoskeletal score
pc_temp	APACHE temperature score	pc_mbps	APACHE mean BP score
pc_crs	APACHE creatinine score	pc_dxs	DX score
male	sex of the patient		

included only if (a) they accurately described the patient's status at the time of hospital admission, (b) they documented conditions occurring frequently enough to be worth collecting, and (c) they were reliably recorded in the medical record. Rand used logistic regression of death within 30 days of admission to build its sickness-at-admission scale, employing backward selection from the full model with all 83 predictors listed in Table 1.1. (The total APACHE II score (Knaus et al. 1985) is a pre-existing 36–point scale that measures sickness for patients in intensive care units; it is the sum of a variety of subscales measuring such things as coma intensity and respiratory rate. Most of the other variables are on 2– to 5–point scales.) They chose (Keeler 1993) the model with 14 predictors shown in Table 1.2.

This is a reasonably good model on grounds of accuracy and parsimony—for instance, its pseudo-R^2 value (Stata 1997) is 28.1% on 14 degrees of freedom, versus 33.4% for the full model with all 83 variables—but no account has been taken in its construction of the *data collection costs* of the variables chosen. Expressed in terms of time for a skilled data collector to abstract the variables from patient medical records, the sickness indicators range from about *10 seconds* to more than *15 minutes* to collect. It is quite possible that a different subset of the predictors in Table 1.1 would be more cost-effective in measuring quality of care in this way than the full list of 14 variables.

1.3 Contents

The plan of the dissertation is as follows.

In this chapter I have given some general details on the health policy background, by presenting the source of the problem and the way that Rand tried to solve it, discussing the way that the variables for each disease were selected, and giving a table of the final admission sickness variables (for pneumonia, the disease I have chosen) that I will be using in the project.

In Chapter 2 I formulate the basic problem more precisely, by giving details on the objectives and purpose of this work and describing the basic strategies I will follow. I present the utility function whose expected value I will maximise, analyse its components, and sketch some of the difficulties that need to be overcome.

In Chapter 3 I give a full description of the optimisation algorithms I will compare in the attempt to maximise the expected utility specified in Chapter 2. I present five different methods—*genetic algorithms* (GA), *messy simulated annealing* (MSA), *simulated annealing* (SA), *tabu search* (TS), and *threshold acceptance* (TA)—analyse their inputs together with the generic and problem-specific choices that need to be

Table 1.2: *The final variables for pneumonia chosen by Rand.*

Name	Meaning
pc_aps2	Total APACHE II score
sage2	Age of patient
p_sbpd1	Systolic blood pressure score
pc_xchfs	CHF chest X-ray score
sbun	Blood urea nitrogen
pc_comas	APACHE II coma score
pc_albms	Serum albumin score
p_sobd1	Shortness of breath day 1
pc_resps	Respiratory distress
septic	Septic complications
pc_prfs	Prior respiratory failure
pc_phsps	Recently hospitalised
pc_ambls	Ambulatory score
ctemp	Initial temperature

made, and discuss modifications to the standard methods required for this particular problem.

Chapter 4 presents results in a special case involving only the $p = 14$ variables used in the Rand scale, where direct examination of all possible models ("full enumeration") suffices to identify the best subsets. In this chapter the geometry of the solution space is also explored. The quantity to be optimised cannot be computed exactly in closed form, so I estimate it by Monte Carlo methods, and I clarify the rôle of N, the number of simulation replications, in the optimisation process. I present some preliminary findings comparing the five optimisation techniques outlined in Chapter 3, in a version of the 14–variable case in which all of the methods are severely constrained on the total CPU time available for the search (no more than 20 minutes of CPU time at 400 Unix MHz), and I then present results from a large simulation experiment to investigate the quality of the solutions from the three main optimisation algorithms—GA, SA, and TS—as a function of the method's inputs. The chapter closes with a variety of sensitivity analyses exploring the robustness of the problem formulation and results.

(The computations in this dissertation were performed on a variety of Unix workstations whose CPU speeds ranged from 100 to 400 Unix MHz. I have standardised all timings so that they are based on 400 Unix MHz. For comparison, Unix MHz is typically 2–3 times faster than PC MHz in the types of calculations employed here.)

In Chapter 5 I present results for the case with all 83 variables, where the space over which I am optimising is vastly larger, continuing the specialisation of the results to the three main methods. Full enumeration is impossible in this situation, so I have created a proxy for the list of the k best models (for $k = 3,000$) by running each of GA, SA, and TS (with the best input settings in the 14–variable case) for a week of CPU time, merging the results, removing duplicate subsets of variables, and performing full enumeration on the 3,000 best models found in this way. I present results for each of the three optimisation methods, in which each method is given a budget of 3 hours of CPU time, using a variety of input settings, and I also provide results in which each method was allowed 24 hours of CPU time. I conclude the chapter with an overall comparison of the optimisation methods. Chapter 6 brings the dissertation to a close with some discussion and comments on future work.

Having outlined what I have done, it would also perhaps be useful to point out one thing I have *not* done. The statistical problem addressed here is variable selection in generalised linear models, a topic which has generated a vast literature and many ad hoc ideas. It is possible to conceive of two distinctly different questions that a dissertation like this one could address:

- How well do some of the leading stochastic optimisation methods perform when they are guided by one or more ad hoc variable selection heuristics?
- How well do such methods perform when they are not guided in this way?

Since in many optimisation problems it is difficult to generate such heuristics, I regard both of these questions as interesting. I have chosen to answer the second question in the work presented here; Chapter 6 gives some ideas for how the first question might be addressed.

A portion of this work (mainly material from Chapter 2 and the preliminary results from Chapter 4) was written up for publication for an optimisation journal late in 1999 and is available in (Draper and Fouskakis 2000). We are now working on three more papers: a review article on stochastic optimisation based on Chapter 3 for a statistical audience, a methodology paper for a statistics journal based on the later results in Chapters 4 and 5 (and some of the ideas for future work in Chapter 6), and an overview paper for a health policy journal.

Chapter 2

Problem formulation

The whole is more than the sum of its parts.

— Aristotle

2.1 Decision-theoretic approach

I have argued in Chapter 1 that the goal in constructing a scale for measuring patient sickness at admission should be to balance data-collection cost against accuracy of prediction of an outcome such as 30-day mortality. When cost and accuracy are weighed against each other, a large *optimisation problem* arises in which expensive variables that do not predict well enough should be omitted from the scale. This optimisation problem cannot be solved by brute-force enumeration: for example, when $p = 83$ it is necessary to compare $2^p \doteq 9.7 \cdot 10^{24}$ subsets of sickness variables, and even at the rate of 100 subsets examined per second—which is far faster than present computational resources permit using desktop workstations—it would take more than $3 \cdot 10^{15}$ years to find the optimal subset by looking at all of them.

Suppose (a) the 30–day mortality outcome y_i and data on p sickness indicators $(x_{i1}, \ldots, x_{ip})$ have been collected on n individuals sampled randomly from a population $\mathcal{P}$ of patients with a given disease, and (b) the goal is to predict the death outcome for m new patients who will in the future be sampled randomly from $\mathcal{P}$, (c) on the basis of some or all of the predictors x_j, when (d) the marginal costs of data collection per patient $c_1, \ldots, c_p$ for the x_j vary considerably. What is the best subset of the x_j to choose, if a fixed amount of money is available for this task and you are rewarded based on the quality of your predictions?

To solve this problem I take a *Bayesian decision-theoretic* approach (Bernardo and Smith 1994) based on *maximisation of expected utility.* Any reasonable utility function here will have two components, one quantifying *data collection costs* associated with the construction of a given sickness scale, the other rewarding and penalising the scale's *predictive successes and failures.*

2.1.1 Data-collection utility

I follow traditional statistical usage and refer to a subset of the x_j as a *model.* One difficulty with the problem statement above is that by definition the future patients are unobserved, but—given that both the present and future samples are randomly drawn from $\mathcal{P}$—a random subsample of the available data will be a good proxy for the future data. Thus to estimate the predictive success of a given model on future patients I use the cross-validation idea (Hadorn et al. 1992) of (1) dividing the available data at random into modelling and validation subsamples M and V, of size n_M and $n_V = n - n_M$ (respectively); (2) fitting the model to the data in M; and (3) evaluating its predictive accuracy on V. In Chapter 4 I present results with the choice $\left(\frac{n_M}{n}, \frac{n_V}{n}\right) = \left(\frac{2}{3}, \frac{1}{3}\right)$; this chapter also contains some results on the sensitivity of the findings to this choice.

In the approach presented here utility is quantified in monetary terms, so that the data collection utility is simply the negative of the total amount of money required to gather data on the specified predictor subset. Letting $I_j = 1$ if x_j is included in a given model (and 0 otherwise), the data-collection utility associated with subset $I = (I_1, \ldots, I_p)$ for patients in the validation subsample is

$$U_D(I) = -n_V \sum_{j=1}^{p} c_j I_j, \tag{2.1}$$

where c_j is the marginal cost per patient of data abstraction for variable j. In the Rand study described in Chapter 1, the data—on which this dissertation is based—consisted of a representative sample of 16,792 elderly American patients hospitalised in the period 1980–86 with one of six high-prevalence diseases. As mentioned above, I focus here on pneumonia, for which the sample size was $n = 2,532$; the marginal costs per variable in this study were obtained by approximating the average amount of time needed by qualified nurses to abstract each variable from medical records and multiplying these times by the mean wage (about US$20 per hour in 1990) for the abstraction personnel.

2.1.2 Predictive utility

To measure the accuracy of a model's predictions, a metric is needed which quantifies the discrepancy between the actual and predicted values, and in our problem this metric must come out in monetary terms on a scale comparable to that employed with the data-collection utility. In the setting of this case study the actual values y_i are binary death indicators and the predicted values $\hat{p}_i$, based on statistical modelling, take the form of estimated death probabilities. I have chosen an approach to the comparison of actual and predicted values that involves dichotomising the $\hat{p}_i$ with respect to a cutoff, to mimic the decision-making reality that actions taken on the basis of input-output quality assessment will have an all-or-nothing character at the hospital level (for example, regulators must decide either to subject or not subject a given hospital to a more detailed, more expensive quality audit based on process criteria). Other, continuous, approaches to the quantification of predictive utility are possible (e.g., a log scoring method (Bernardo and Smith 1994)); I intend to explore this in future sensitivity analyses (not presented in this dissertation).

In the first step of the approach taken here, given a particular predictor subset I, I fit a logistic regression model to the modelling subsample M and apply this model to the validation subsample V to create predicted death probabilities $\hat{p}_i^I$. In more detail, letting $y_i = 1$ if patient i dies and 0 otherwise, and taking $x_{i1}, \ldots, x_{ik}$ to be the k sickness predictors for this patient under model I, the statistical assumptions underlying logistic regression in this case are

$$\begin{array}{c} (y_i \,|\, p_i^I) \overset{\text{indep}}{\sim} \text{Bernoulli}(p_i^I), \\ \log(\frac{p_i^I}{1-p_i^I}) = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}. \end{array} \tag{2.2}$$

I use maximum likelihood to fit this model, obtaining a vector $\hat{\beta}$ of estimated logistic regression coefficients, from which the predicted death probabilities for the patients in subsample V are given by

$$\hat{p}_i^I = \left[1 + \exp\left(-\sum_{j=0}^{k} \hat{\beta}_j x_{ij} \right) \right]^{-1}, \tag{2.3}$$

where $x_{i0} = 1$ ($\hat{p}_i^I$ may be thought of as the sickness score for patient i under model I). (If any of the predictors x_j are highly collinear, this problem should be solved in one of the usual ways (Weisberg 1985) before fitting model (2.2); for example, if a pair of predictors is highly correlated one of them could be dropped, or a scale could be created from the two of them using principal components (Chatfield and Collins 1980). This problem does not arise with the Rand data.)

In the second step of the approach taken here, I classify patient i in the validation subsample as predicted dead or alive according to whether $\hat{p}_i^I$ exceeds or falls short

Table 2.1: *Cross-tabulation of actual versus predicted death status. The left-hand table records the monetary rewards and penalties for correct and incorrect predictions; the right-hand table summarises the frequencies in the* 2×2 *tabulation.*

		Rewards and Penalties		Counts	
		Predicted		Predicted	
		Died	Lived	Died	Lived
Actual	Died	C_{11}	C_{12}	n_{11}	n_{12}
	Lived	C_{21}	C_{22}	n_{21}	n_{22}

of a cutoff p^*, which is chosen—by searching on a discrete grid from 0.01 to 0.99 by steps of 0.01—to maximise the predictive accuracy of model I. I then cross-tabulate actual versus predicted death status in a 2×2 contingency table, rewarding and penalising model I according to the numbers of patients in the validation sample which fall into the cells of the right-hand part of Table 2.1. (To clarify the rôle of the probability cutoff, for each of the 99 values of p^* from 0.01 to 0.99 I calculated the entries in Table 2.1 and the resulting predictive utilities in equation (2.4), and then chose the cutoff p^* which maximises this utility. In practice the optimal cutoff was typically around 0.4.) The left-hand part of this table records the rewards and penalties in US$. The predictive utility of model I is then

$$U_P(I) = \sum_{l=1}^{2} \sum_{m=1}^{2} C_{lm} \, n_{lm}. \tag{2.4}$$

The following process was used to elicit the utility values C_{lm}. Clearly C_{11} and C_{22} should be positive and C_{12} and C_{21} negative, and since it is easier to correctly predict that a person lives than dies with these data (the overall pneumonia 30–day death rate in our sample was 16%, so if you predict that every patient lives you will be right about 84% of the time) it is natural to choose the C_{lm} so that $C_{11} > C_{22}$. It is also clear from the fact that it is worse to label a "bad" hospital as "good" than the other way around that one should take $|C_{12}| > |C_{21}|$, and furthermore that the magnitudes of the penalties should exceed those of the rewards. It seemed natural to specify the C_{lm} by eliciting two kinds of information from health experts in the US and UK: one of the four values, say C_{12}, and the ratios of the other three C_{lm} to this value.

Since the utility structure used here is based on the idea that hospitals have to be treated in an all-or-nothing way in acting on the basis of their apparent quality, the approach taken was (i) to attempt to quantify the monetary loss L of incorrectly subjecting a "good" hospital to a detailed but unnecessary process audit and then (ii) to translate this from the hospital to the patient level. A rough correspondence may be made between the left-hand part of Table 2.1 at the patient level and a hospital-level table with rows representing truth ("bad" in row 1, "good" in row 2) and columns representing the decision taken ("process audit" in column 1, "no process audit" in column 2). Unnecessary process audits then correspond to cell $(2, 1)$ in these tables (hospitals where a process audit is not needed will typically have an excess of patients who are predicted to die but actually live). Discussions with health experts in the US and UK suggested that detailed process audits cost on the order of $L =$ US\$5,000, and Rand data indicated that the mean number of pneumonia patients per hospital per year in the US at the time of the Rand PPS quality of care study was 71.8. This fixed C_{21} at approximately $\frac{-\$5{,}000}{71.8} = -\$69.6$. The health experts judged that C_{12} should be the largest in absolute value of the C_{lm}, and—averaging across the expert opinions, expressed as orders of magnitude base 2—the elicitation results were $\left|\frac{C_{12}}{C_{21}}\right| = 2$, $\left|\frac{C_{11}}{C_{21}}\right| = \frac{1}{2}$, and $\left|\frac{C_{22}}{C_{21}}\right| = \frac{1}{8}$, finally yielding $(C_{11}, C_{12}, C_{21}, C_{22}) = \$(34.8, -139.2, -69.6, 8.7)$. The results in Chapters 4 and 5 below use these values; in Chapter 4 I also present a sensitivity analysis on the choice of the C_{lm}.

Total expected utility. The overall expected utility function to be maximised over I is then simply

$$E\left[U(I)\right] = E\left[U_D(I) + U_P(I)\right]. \tag{2.5}$$

In practice I use Monte Carlo methods to evaluate this expectation, averaging over N random modelling and validation splits. The optimal choice of N is an important practical problem which I will address in Chapter 4.

2.2 The goals of this project

With p predictors to choose from, the expected utility maximisation is over 2^p possible subsets of variables. With the data described here it takes about 0.4 seconds on a Sun `UltraSPARC` Enterprise 250 computer running Unix at 400Mhz to evaluate $E\left[U(I)\right]$ for a single modelling/validation split with efficient code and $p = 14$, so (as mentioned in Chapter 1) it is computationally infeasible given present computing resources—even with a moderate choice of N—to perform exhaustive enumeration for all $p = 83$ sickness indicators for pneumonia. Attention thus naturally focuses on stochastic optimisation as a way to find "good" (near-optimal) subsets for large p.

In Chapters 4 and 5 I compare the usefulness of methods of stochastic optimisation based on Markov Chain Monte Carlo, including simulated annealing (SA (Kirkpatrick et al. 1983)), and competitors such as genetic algorithms (GA (Holland 1975)), tabu search (TS (Glover 1989)), messy simulated annealing (MSA (Kvasnička and Pospíchal 1995), and threshold acceptance (TA (Dueck and Scheuer 1990)) in solving this large optimisation problem. I use the case $p = 14$ for pneumonia and for the predictors shown in Table 1.2 as a particularly valuable testbed. By performing exhaustive enumeration with this setup to find the global mode and a number of other apparently promising local modes, I can then try out various optimisation strategies with a relatively small amount of search time to get an idea of what will work best with $p = 83$. This work first of all provides a new perspective on variable selection in generalised linear models and also offers new insights into the comparative advantages and flaws of competing optimisation methods. Finally I hope it will produce results of direct use in health policy.

Throughout the results of the simulation experiments presented here I have taken the point of view that the only fair and practical way to compare optimisation methods is to give each of them a fixed budget of CPU time, as opposed to a fixed number of models (input configurations) visited. My reason for this choice is as follows. The user of an optimisation algorithm has only her/his computer sitting in front of her/him and a fixed budget of time in which to solve the current problem, with other problems waiting to be solved in the future—in other words, spending too much time on this problem has a real cost in terms of being able to spend less time on future problems. From this viewpoint the only thing that matters is how well any method performs with a fixed budget of CPU time. Suppose, for example, that I have two optimisation methods, one of which can visit one model per second and the other one model per day (because of huge overhead costs in maintaining its internal algorithmic structure). If I give them both a budget of 1,000 models, it might well be that the second one finds, say, a 10% better set of models, but the first one obtains its results in less than 17 minutes and the other one requires almost three years. It seems clear to me which one I would use.

Chapter 3

Stochastic optimisation

Theory attracts practice as the magnet attracts iron.

— Karl Friedrich Gauss

3.1 Introduction

In the past 40 years many researchers have studied the problem of optimising an objective function. One approach is *stochastic optimisation*, in which the search for the optimal solution involves randomness in some way. In this dissertation I consider a class of problems with a combinatorial nature, where the variables are discrete. The problem of finding the optimal solution in this case is known as *combinatorial optimisation.* If S denotes the finite set of all possible solutions, my task is to maximise or minimise the objective function $f \colon S \rightarrow \Re$. In the case of maximisation, the problem is to find a solution $i_{opt} \in S$ which satisfies

$$f(i_{opt}) \geq f(i) \quad \text{for all } i \in S. \tag{3.1}$$

It is easy to see that as the dimension of S increases the harder the task becomes, and more time is needed to find the optimal, or at least a near–optimal, solution. Another difficulty in this problem is the possibility of *local optima.* It is a usual phenomenon for the objective function to have many local optima. So an algorithm like the well-known *local search*, which only accepts moves with higher values of the objective function than the previous move, is not a very good idea for this situation, since it is likely that the search will get stuck in a local optimum.

Algorithm 3.1. LOCAL SEARCH:

- *Begin;*

```
•   Choose a random configuration $i_{start}$;
•   Set $i := i_{start}$;
•   Repeat:
•     Generate a new configuration $j$ from the neighbourhood of $i$;
•     If $f(j) \geq f(i)$ then $i := j$;
•   Until $f(j) \leq f(i)$ for all $j$ from the neighbourhood of $i$;
• End.
```

□

The disadvantages of local search algorithms can be formulated as follows:

- By definition, local search algorithms terminate in a local maximum and there is generally no information as to the amount by which this local maximum deviates from a global maximum;

- The obtained local maximum depends on the initial configuration, for the choice of which generally no guidelines are available; and

- In general, it is not possible to give an upper bound for the computation time.

To avoid some of the above mentioned disadvantages, one might think of a number of alternative approaches:

- Execution of the algorithm for a large number of initial configurations, say M, at the cost of an increase in computation time; for $M \to \infty$, such an algorithm finds a global maximum with probability 1, if only for the fact that a global maximum is encountered as an initial configuration with probability 1 as $M \to \infty$;

- Use of information gained from previous runs of the algorithm to improve the choice of an initial configuration for the next run;

- Introduction of a more complex generation mechanism, in order to be able to "jump out" of the local maxima corresponding to the simple generation mechanism. To choose the more complex generation mechanism properly requires detailed knowledge of the problem itself; and

- Acceptance of moves which correspond to a decrease in the objective function in a limited way.

In this dissertation I use three well-known methods (plus several variations on them), each of which—because of its structure—manages to avoid the disadvantages of local search algorithms. My goal is to find out which of these three methods performs best in the problem posed in Chapter 2. In this Chapter I briefly analyse these methods, give the exact algorithms, the advantages and disadvantages for each one, and I also summarise the literature on optimal values of the inputs each of the algorithms uses.

3.2 Simulated annealing (SA)

The use of *simulated annealing* (SA) (Kirkpatrick et al. 1983) as a technique for discrete optimisation dates back to the early 1980s. It was heralded with much enthusiasm as it appeared to be both simple to implement and widely applicable, and as a result of articles in popular scientific journals researchers from a wide variety of disciplines experimented with it in the solution of their own problems.

The ideas that form the basis of SA were first published by (Metropolis et al. 1953) in an algorithm to simulate the cooling of material in a heat bath—a process known as annealing. If solid material is heated past its melting point and then cooled back into a solid state, the structural properties of the cooled solid depend on the rate of cooling. The annealing process can be simulated by regarding the material as a system of particles. Essentially, the Metropolis algorithm simulates the change in energy of the system when subjected to a cooling process, until it converges to a steady "frozen" state. Thirty years later (Kirkpatrick et al. 1983) suggested that this type of simulation could be used to solve optimisation problems.

SA is a *stochastic local search* technique to approximate the maximum of the objective function $f\colon S \to \Re$ over a finite set S. It is an iterative method that randomly chooses elements y from a neighbourhood $N(x)$ of the present solution. The candidate y is either accepted as the new solution or rejected. It may be accepted with a positive probability even if $f(y) < f(x)$, that is, even if it is worse than the present solution. The search process can thus "climb uphill" to get out local maxima. SA has proven quite successful in many applications (Van Laarhoven and Aarts 1988), and thus anyone considering the use of SA today has access to a wide range of literature covering both theoretical and empirical results.

The long-run behaviour of the search process depends critically on $a(x, T, y)$, the probability of accepting a candidate y given a present solution x. $a(x, T, y)$ is controlled by the parameter T, which is called the *temperature* by analogy to a physical cooling process. To make the iterative search an inhomogeneous Markov Chain, the temperature values are chosen independently of the process as a fixed sequence T_n, the *temperature schedule.* Usually the *Metropolis acceptance probability* is used; that is, for $T > 0$,

$$a(x, T, y) := \left\{ \begin{array}{ll} 1 & \text{if } f(y) \geq f(x) \\ \exp\left[\frac{f(y)-f(x)}{T}\right] & \text{if } f(y) < f(x) \end{array} \right\}. \tag{3.2}$$

From the Metropolis acceptance probability you can see that the better moves are always accepted, but on the other hand there is a possibility to accept a move to a worse solution than the present solution with probability $\exp\left[\frac{f(y)-f(x)}{T}\right]$. At high temperatures, the system accepts moves almost randomly, regardless of whether they are uphill or down. As the temperature is lowered, the probability of accepting downhill moves drops and the probability of accepting uphill moves rises. Eventually the system "freezes" in a locally or globally maximum state, and no further moves are accepted. The rate at which T decreases as the number of iterations increases is crucial. I will speak later about the temperature schedules that I will mostly be using.

The candidate moves are chosen according to a generating probability $G(x, \cdot)$, which is often the *uniform* or *normal* distribution on the neighbourhood $N(x)$. The algorithm can be stated as follows:

Algorithm 3.2. SIMULATED ANNEALING (SA):

```
• Begin;
•     Choose a configuration $i_{start}$;
•     Select the initial and final temperatures $T_0, T_f > 0$;
•     Select the temperature schedule;
•     Set $i := i_{start}$ and $T := T_0$;
•     Repeat:
•       Repeat:
•         Choose a new configuration $j$ from the neighbourhood of $i$;
•         If $f(j) \geq f(i)$ then $i := j$;
•         Else
•           Choose a random $u$ uniformly in the range $(0,1)$;
•           If $u < \exp\left[\frac{f(j)-f(i)}{T}\right]$ then $i := j$, else $i := i$;
•       Until iteration count $= n_{iter}$;
•       Decrease $T$ according to the temperature schedule;
•     Until stopping criterion = true;
•     $i$ is the approximation to the optimal solution;
```

- *End.*

□

The final configuration at the end of the SA run can be reported as the approximate solution, or the k best solutions found so far (for some reasonable $k \geq 1$) can be maintained throughout the run and reported (this requires some CPU and memory resources to implement but is often worthwhile).

It is worth noting that there is a substantial difference in outlook about SA between the statistics and operational research (OR) communities. In statistics the objective function f is usually a density function, $G(x, \cdot)$ is called the *proposal distribution*, and SA is viewed as a Metropolis-Hastings algorithm (Aarts and Korst 1989) on a series of heated densities. By contrast, in the optimisation literature f can be any function whose global maximum is sought, and the precise forms of the generating probability $G(x, \cdot)$ and the acceptance probability are typically chosen from much larger sets of possibilities (see Sections 3.2.1 and 3.2.2 below). See (Aarts and Korst 1989; Geman and Geman 1984) for convergence results to local or global optima for SA whether or not f is a density.

The algorithm given above is very general, and a number of decisions must be made in order to implement it for the solution of a particular problem. These can be divided into two categories. Firstly there are generic decisions which are concerned with parameters of the annealing algorithm itself. These include factors such as the initial temperature, the *cooling schedule*, the parameter n_{iter}, and the stopping criterion. The second class of decisions is problem-specific and involves the choice of the space of feasible solutions, the form of the objective function and the neighbourhood structure employed.

Both types of decisions need to be made with care, as they have been shown to affect the speed of the algorithm and the quality of the solutions obtained. There has been much research into the theoretical convergence properties of the annealing algorithm. This work does provide pointers as to what factors should be considered in making both generic and problem-specific decisions, but of course these choices depend on the nature of the problem you are trying to solve.

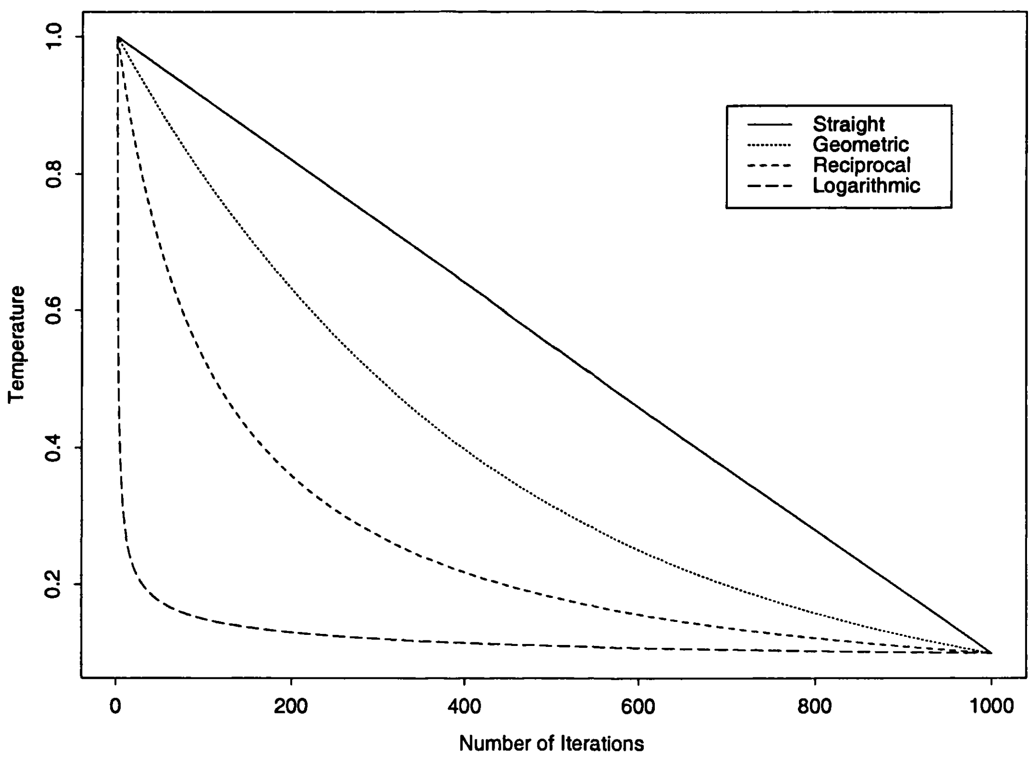
3.2.1 Generic and problem-specific decisions

The generic decisions basically involve the cooling schedule, including the upper and lower limits for the temperature parameter and the rate at which it must be reduced. The two cooling schedules which occur most widely in practice illustrate opposite

Table 3.1: *Families of temperature schedules for simulated annealing.*

Family	Temperature T_i
Straight	$\frac{T_f - T_0}{M-1}(i-1) + T_0$
Geometric	$T_0 \left(\frac{T_f}{T_0}\right)^{\frac{i-1}{M-1}}$
Reciprocal	$\frac{T_0 T_f (M-1)}{(T_f M - T_0) + (T_0 - T_f) i}$
Logarithmic	$\frac{T_0 T_f [\log(M+1) - \log 2]}{T_f \log(M+1) - T_0 \log 2 + (T_0 - T_f) \log(i+1)}$

Figure 3-1: *The four temperature schedules in Table 3.1, with $T_0 = 1.0, T_f = 0.1$, and $M = 1,000$.*



extremes. The first is the most commonly used and involves *geometric* reduction:

$$T_{new} = T_{old}\,(1 - \epsilon). \tag{3.3}$$

Experience (Stander and Silverman 1994) has shown that relatively small values of ϵ perform best and most reported successes in the literature use values between 0.2 and 0.01. This corresponds to fairly slow cooling. You can also define ϵ, subject to the upper and lower limits for the temperature parameter and the final number of iterations M. So if T_0 and T_f are the initial and final limits for the temperature, then you can take ϵ to be

$$\epsilon = 1 - \left(\frac{T_f}{T_0}\right)^{M^{-1}} . \tag{3.4}$$

Another parameter is the number of iterations at each temperature, n_{iter}. This is usually related to the size of the neighbourhoods, or sometimes the solution space, and may vary from temperature to temperature. For example, it is important to spend a long time at lower temperatures to ensure that a local optimum has been fully explored. This can be done by increasing the value of n_{iter} either geometrically (by multiplying by a factor greater than one) or arithmetically (by adding a constant factor) at each new temperature. Also you can determine n_{iter} based on feedback from the process. For example it may be desirable to accept a certain number of moves before decreasing the temperature. So you will probably have to spend a very short amount of time at high temperatures when the acceptance rate is high, but on the other hand it may take an infeasible amount of time to reach the required total number of accepted moves in the case that temperature is low and the acceptance rate is very small.

Another commonly used schedule, suggested by (Lundy and Mees 1986), executes just one iteration at each temperature, but reduces temperature very slowly according to the formula

$$T_{new} = \frac{T_{old}}{1 + \beta T_{old}}, \tag{3.5}$$

where β is a suitable small value. You can of course easily define β, subject to the upper and lower limits for the temperature parameter and the final number of iterations M.

A large number of cooling rates have been proposed in the literature. In Table 3.1 I show the most common ones—*straight, geometric, reciprocal,* and *logarithmic*—indexed by the initial temperature T_0, the final temperature T_f, the run-length M and the current iteration i, and Figure 3–1 plots these four schedules with $T_0 = 1.0, T_f = 0.1$, and $M = 1,000$. Both empirical evidence and the theoretical research suggest that the precise shape of the family of cooling schedules is less important than the amount of time SA spends in high, medium, and low temperature ranges. So there is little to choose between, say, the geometric and Lundy-Mees schedules, as long as they cool over the same range of temperatures at approximately the same rate. In view of this result and those of experiments reported in the literature, when using annealing for a new application it is probably best to start off with one of these two schedules first and only consider the others if these fail to provide satisfactory results. In terms of deciding on the values of the parameters for the schedule chosen, there is no easy way of achieving this; usually the best parameters must be determined after much experimentation, and of course they are subject to the nature of the problem you are solving.

On the problem-specific decisions, as with the generic decisions it is not possible to set down a series of rules which will always define the best choice for a given problem. However, it is possible to outline some properties which are desirable. Firstly concerning the neighbourhood structure, it is good to be uniform and symmetric, so that all the solutions have the same number of neighbours, and also if i is a neighbour of j, then j should of course be a neighbour of i as well. If you want to keep the computing time as low as possible it is important for the neighbourhood structure and the cost function to be chosen in such a way that the calculations to be made in every iteration can be carried out quickly and efficiently. So it is good if the neighbourhoods are not large and complex, and if the solution space is not constrained by strict feasibility conditions. Also the cost function is going to lead the process towards local maxima, and so large plateau-like areas where the cost function takes on equal values should be avoided. To keep the working solution space small, it may be useful to try to have reasonably small neighbourhoods. This enables a neighbourhood to be searched adequately in fewer iterations, but conversely means that there is less opportunity for dramatic improvements to occur in a single move. Thus there must be some compromise here but, in general, small simple neighbourhoods are preferable to large complex ones.

3.2.2 Modifications

In this section I examine a number of modifications which have proved useful in adapting the annealing algorithm for a number of different problems. It is worth mentioning here that these modifications appear in the literature in only a few examples, so it is better to consider them only in situations where the annealing algorithm described so far fails to provide satisfactory results.

- **Acceptance probability**

Firstly consider changing the acceptance probability. The use of the *Boltzmann distribution* (Tipler 1969) in (3.2) arises entirely from the laws of thermodynamics, and there is no reason to suppose that some other distribution would not perform better in some specific examples. On the other hand the use of the Boltzmann distribution has the advantage that it accepts downhill moves in such a way that large declines in f have virtually no chance of acceptance, whereas small ones may be accepted regularly.

The biggest problem with the Boltzmann distribution is the algorithm speed. The calculation of $\exp\left[\frac{f(y)-f(x)}{T}\right]$ at every iteration is quite a time-consuming procedure, and so it might be better to evaluate a cheaper function. (Johnson et al. 1989) suggests two possible methods of improvement. The first is to use the function

$$a(x,T,y) := \left\{ \begin{array}{ll} 1 & \text{if } f(y) \geq f(x) \\ 1 + \frac{f(y)-f(x)}{T} & \text{if } f(y) < f(x) \end{array} \right\}, \tag{3.6}$$

which approximates the exponential (but note that this probability could go negative). Even better than this is to use a discrete approximation represented by a look-up table which can be calculated at a series of fixed values over the range of possible values of $\frac{f(y)-f(x)}{T}$. The approximation is then obtained by simply rounding $\frac{f(y)-f(x)}{T}$ to the nearest integer and looking up the appropriate function value. Finally there are a few researchers that have found that simpler functions can give good results. For example (Brandimarte et al. 1987) uses the form

$$a(x,T,y) := \left\{ \begin{array}{ll} 1 & \text{if } f(y) \geq f(x) \\ -\frac{f(y)-f(x)}{T} & \text{if } f(y) < f(x) \end{array} \right\} \tag{3.7}$$

(note again, however, that this probability can exceed 1); (Ogbu and Smith 1990) and (Vakharia and Chang 1990) both use probabilities which are independent of $[f(y) - f(x)]$, for different sequencing problems.

- **Cooling**

Consider now the case of different cooling schedules. Starting the process with temperatures so high that almost all moves are accepted simply produces a series of random solutions, each one of which might itself have been a starting solution. So one may think that this approach spends too much time with random solutions, making many non-useful and time consuming evaluations. Some researchers address this problem by doing a very rapid cooling phase. They achieve this by reducing temperature after a fixed number of acceptances, and so they use most of the time in the middle part of the temperature range at which the rate of acceptance is relatively small. (Connoly 1990) was the first to suggest a constant temperature approach. Such a temperature must obviously be high enough to allow the process to climb out of local optima, but cool enough to ensure that these local optima are visited. The problem with this approach is that a good temperature will not only vary from problem type to problem type, but will vary for different instances of the same problem. Finally (Downsland 1993) suggests a quite clever idea. Every time a move is accepted the system cools according to the function $T \rightarrow \frac{T}{(1+\beta T)}$, and every time a move is rejected the system is heated according to the function $T \rightarrow \frac{T}{(1-\gamma T)}$. If $\beta = k\gamma$ then the system will need to go through k heating iterations to balance one cooling. If the ratio of rejected moves to accepted moves is greater than k the system heats up, if less the system cools. Thus this schedule theoretically tends to converge to a situation in which the ratio is about k. As it is important to adequately search the areas close to local optima without a significant temperature increase, it is suggested that k should be governed by the size of the neighbourhoods around these maxima.

- **Neighbourhoods**

In the annealing algorithms that have been described so far we are using the assumption that the neighbourhood structure is well-defined and unchanging throughout the algorithm. But this may not always be the case. We might have improvements in the performance of the algorithm if we adjust the neighbourhood structure as the temperature decreases. How can we achieve this? One way is to put restrictions on the neighbourhood in some manner. An example is given in an annealing heuristic for the placement phase of very large scale integrated circuit (VLSI) design, in which rectangular blocks are placed on the chip area in such a way as to minimise a combination of cost factors, described by (Sechen et al. 1988). They include the horizontal and vertical translations of any block in the set of valid neighbourhood moves. As only small translations tend to be accepted at low temperatures, much time is wasted generating and rejecting longer translations. In order to avoid this, a limit on the maximum translation length is imposed and this is decreased as the temperature drops.

In situations where a penalty function is used to enforce constraints, the neighbourhood size can be decreased by allowing only moves involving variables which contribute to the violation of constraints. Finally (Tovey 1988) suggests that better performance may be achieved if a reduced neighbourhood is used with a fixed probability, and the full neighbourhood is used for the remaining iterations.

- **Sampling**

The standard annealing process samples randomly from the current neighbourhood of solutions. One problem with this approach may be that when the process is very close to a local optimum most of the neighbouring solutions will involve a decrease in the criterion function. So by doing random sampling it is quite possible to accept some downhill moves before reaching the local optimum, and thus we may never reach it. One solution to this problem is to make cyclic rather than random sampling, to ensure that all neighbours are tried once before any are considered for a second time. This also has the added advantage of avoiding the need to determine a random neighbour. It should however be noted that there are also reports of cyclic sampling having a negative effect on the solution quality for some annealing implementations.

Another possible problem that has been reported occurs at the end of the algorithm. In this stage the system is relatively cool and much time is spent evaluating moves which are rejected. This can be avoided by determining the acceptance probability for each move in the neighbourhood, sampling the neighbourhood using a weighted distribution given by these probabilities and accepting automatically.

- **The objective function**

There are problems in which the difference in the objective function between the current and the new solution is not calculated quickly. That makes the whole algorithm very slow and inconvenient. (Tovey 1988) suggests that in these cases an approximation may be a good idea. It is possible then to obtain some good results with an objective function which does not precisely represent the true function. If the true objective is evaluated only for each accepted move then the true maximum out of all configurations visited can be retained.

The importance of the objective function in the annealing process has already been discussed. We know that if the system is at a saddle-point between two valleys then it will move to either with equal probability. If one choice leads to the global maximum and the other to a local optimum, a move in the wrong direction may never be recovered. Sometimes this is unavoidable, but in other cases a change in the form of the objective may highlight the one direction as an improvement and the other as a downhill move.

So you can see how important it is to use a "nice" objective function. The situation described above was encountered by (Downsland 1993) for the rectangle problem. In this case a relaxation of the objective function identified the correct move as uphill and the inferior move as downhill. But the problem was that the new relaxed objective function was insufficient by itself to converge to "feasible" solutions with no overlap. In order to use properties of both objective functions you can express the criterion function as a weighted sum of the actual and the relaxed. The temperature parameter will be the same for both acceptance functions, except for the fact that it must be multiplied by different weighting factors for the two different cost functions. Also the acceptance functions may be different; in particular the best results were achieved when the acceptance probability distributions were different—one being exponential and the other linear.

The problem with this method arises when the objective function involves a penalty factor, so that it is very difficult to determine the correct weighting factors for the different terms. In addition, when the true objective function takes on relatively few integers values, it is often necessary to use other cost elements to guide the annealing process across the resulting plateau-like areas. One way of avoiding these difficulties is to solve the problem iteratively, trying to attain feasibility for increasing values of the true objective function. In this way only the penalty function is involved in the annealing cost, and thus no weighting decisions are required. The solution space may also be reduced as it contains only those solutions which achieve the current constant true cost.

The main disadvantage with this approach is that a globally optimal solution with a cost value of zero must be found at each stage in order to attain feasibility. In many situations this may be expecting too much of the annealing algorithm.

- **Combination with other methods**

Many researchers have noted that SA can perform better if it is used in combination with other heuristic methods. In general this can be done by running these methods before the annealing process is invoked (or after, in order to make an improvement to the solution encountered by SA). However there are examples of heuristics being used as a part of the annealing algorithm.

The most common is the use of a pre–processing heuristic in order to determine a good starting solution for our algorithm. To do this we have to start our search at a low temperature, because if we start it at a high one all the characteristics of the good solution will be destroyed. Thus we have the advantage that we save a substantial amount of solution time. However this method may get us caught in a trap, since by starting with a good solution at low temperature the process may never fully escape from the neighbourhood of the starting value.

Another way of incorporating some knowledge into the starting solution is to pre-define some of its features. These could theoretically be destroyed during the annealing process but this is unlikely at lower temperatures.

(Chams et al. 1987) used an approach in their colouring algorithm where they incorporate annealing into a construction heuristic which works by building onto a previous partial solution.

The use of a post-processing heuristic is also quite common in order to ensure that at least a local maximum has been found. Some researchers suggest applying the ascent phase more frequently, but it is difficult to determine when this should be done. One extreme idea is to apply it after every accepted move.

- **Parallel implementations: speeding up the algorithm**

One of the major disadvantages of SA is that application of the algorithm may require large amounts of computation time. Therefore, it is worthwhile investigating possibilities of speeding up the algorithm, in order to keep computation time within reasonable limits. In this respect, the increasing availability of *parallel machines* offers an interesting opportunity to explore the possibilities of speeding up the SA algorithm. That's why research on *parallel implementations of SA* has evolved so quickly in recent years. The key idea in designing parallel SA algorithms is to distribute the execution of the various parts of SA over a number of communicating parallel processors.

(Aarts and Korst 1989) identify three ways in which parallelism may be introduced into the annealing process. The most common and simplest way is to allow different processors to proceed with annealing using different streams of random numbers, until the temperature is about to be reduced. Then, the best result from all the processors is chosen and all processors start again from this common solution at the new temperature. With this method, when the temperature is high, we expect to have significantly different chains of solutions among the different processors, but when the temperature drops to low values we expect that the processors will end up with solutions very close in terms of neighbourhood structure and cost.

A second method of parallel implementation is to use the processors to generate random neighbours and test for acceptance independently. Once a processor finds a neighbour to accept, then this is conveyed to all the other processors and the search moves to the neighbourhood of the new current solution. Again with this strategy we will have almost all the solutions accepted at high temperatures and so you can say that the method at this point is wasteful, but as temperature decreases the majority of solutions will be rejected and this will speed up the search considerably. You can also consider using the two strategies just described together. You can start with the first method, until the ratio of rejections to acceptances exceeds a certain level and then switch to the second method.

Finally a third (but less common) method is to hold the current solution in common memory and to allow all processors to act on it independently, each one generating a neighbour and updating. However, it is possible that this will result in two moves being made which both give an improvement when considered independently, but result in a downhill move if they are both carried out.

Before closing this section about parallel SA as an attempt to attain greater speed, I briefly discuss two other alternative approaches that speed up the algorithm.

3.2.3 Messy simulated annealing (MSA)

In this section I introduce an idea proposed by (Kvasnička and Pospíchal 1995), where the original SA method is modified using ideas borrowed from the genetic algorithms literature (see Section 3.4 below). Consider a function $f(X)$, with variables $X_1, X_2, \ldots, X_p$, where $X_i = 0$ or 1. We call $X_1, X_2, \ldots, X_p$ the *genes* of the *chromosome* X and their binary values *alleles.* Our aim is to find the maximum of f.

Let $Q = \{1, 2, \ldots, p\} \times \{0, 1\}$ be a set which contains all possible pairs (α, β), where $\alpha \in \{1, 2, \ldots, p\}$ is the *gene name* and $\beta \in \{0, 1\}$ is the *allele* of gene α. Then *M-chromosomes* of length l are defined by

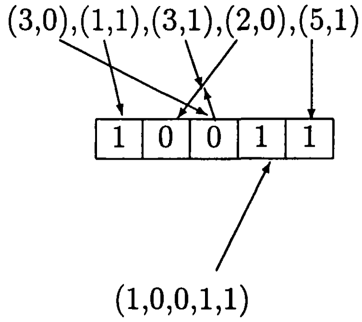
$$c = [(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_l, \beta_l)] \in Q^l. \tag{3.8}$$

The first pair specifies the allele of the α_1 gene, i.e. $X_{\alpha_1} = \beta_1$. The second pair (α_2, β_2), if $\alpha_1 \neq \alpha_2$, specifies gene α_2, i.e., $X_{\alpha_2} = \beta_2$. In general we have

$$X_{\alpha_i} = \beta_i \quad \text{if} \quad \alpha_i \neq \alpha_j \quad \text{for all} \quad j \leq i - 1. \tag{3.9}$$

This means that the possible difficulty of overspecification of the genes with respect to the target problem is handled by a first-come-first-served rule on a left-to-right scan of the M–chromosome. There is still a possibility not all genes will be specified

Figure 3-2: *An example of decoding of the M-chromosome* $c = [(3,0),(1,1),(3,1),(2,0),(5,1)]$ *to the chromosome* $(1,0,0,1,1)$ *with respect to the template* $(1,0,0,1,1)$.



after the scanning procedure. Then the non-specified ones are filled in by a *template*

$$t = (t_1, t_2, \ldots, t_p) \in \{0,1\}^p. \qquad (3.10)$$

So if the q^{th} gene is not specified, i.e., $q \neq \alpha_i$ for all $i \leq l$, then its allele is equal to the q^{th} entry in the template t, i.e., $X_q = t_q$.

To understand the above more clearly here is an example (see Figure 3–2). Consider $p = l = 5$, the M–chromosome $c = [(3,0),(1,1),(3,1),(2,0),(5,1)]$ and the template $t = (1,0,0,1,1)$. The chromosome X assigned to c by the decoding procedure is then $X = (1,0,0,1,1)$. We assign the allele 0 for X_3 because it appears before the allele 1 in the M–chromosome, and for X_4, which is not specified by c, we use the fourth entry of the template t.

We use two kinds of transformation between the M–chromosomes (Figure 3–3). First is the *allelic mutation*, where we change the alleles from 0 to 1 or from 1 to 0 with probability p_{allele}. Formally if we have the M–chromosome $c = [(\alpha_1,\beta_1),(\alpha_2,\beta_2),\ldots,(\alpha_l,\beta_l)]$ we keep β_i the same if $r > p_{allele}$, and we change β_i to its binary complement if $r \leq p_{allele}$, where $0 \leq r < 1$ is a uniformly distributed random number. Similarly we have the *genic mutation* where α_i is kept the same if $r > p_{gene}$, and $\alpha_i = \alpha_{i'}$ if $r \leq p_{gene}$, where $1 \leq \alpha_{i'} \leq k$ is a uniformly distributed random integer, selected so that $\alpha_{i'} \neq \alpha_i$. For p_{allele} and p_{gene} it is best to use quite small values like 0.05.

We can now use all the above ideas to construct a new SA algorithm. The idea of the M–chromosomes can be applied in place of the proportional distribution. So instead of doing manipulations with chromosomes $X \in \{0,1\}^p$, we use M–chromosomes $c \in Q^l$, where l is approximately equal to p. The algorithm can

Figure 3-3: *Example of mutation of the M-chromosome* $c = [(3,0),(1,1),(3,1),(2,0),(5,1)]$. *Vertical arrows indicate places where allelic and genic mutations are applied. Allelic mutation switches the bit, while genic mutation randomly changes the position of the bit.*

$$\begin{array}{lllll}
\text{allelic} & & & \text{genic} & \\
\text{mutation} & & & \text{mutation} & \\
\downarrow & & & \downarrow & \\
(3,0), & (1,1), & (3,1), & (2,0), & (5,1) \xrightarrow{mutation} (3,1),(1,1),(3,1),(5,0),(5,1)
\end{array}$$

be stated as follows:

Algorithm 3.3. Messy Simulated Annealing (MSA):

```
• begin
•   Choose an M-chromosome $c_{start}$ and a template $t_{start}$;
•   Select an initial temperature $T_0 > 0$;
•   Select the temperature schedule;
•   Decode the M-chromosome $c_{start}$ to the chromosome $X_{start}$;
•   $c := c_{start}$, $X := X_{start}$, $T := T_0$ and $t := t_{start}$;
•   Repeat:
•     Repeat:
•       For $i = 1, \dots, l$;
•         Generate random $r_1$ uniformly in the range (0,1);
•         Change the gene $i$ of $c$ if $r_1 \leq p_{gene}$ to its binary complement;
•       For $i = 1, \dots, l$;
•         Generate random $r_2$ uniformly in the range (0,1);
•       If $r_2 \leq p_{allele}$ generate a uniform random integer in the range $[1,p]$ which
          is not equal to allele $i$ and replace the allele $i$ by this number;
•       Call the new M-chromosome $c_{proposal}$;
•       Decode the M-chromosome $c_{proposal}$ to the chromosome $X_{proposal}$;
•       If $f(X_{proposal}) \geq f(X)$ then $X := X_{proposal}$ and $c := c_{proposal}$;
•       Else
•         Generate random $u$ uniformly in the range (0,1);
•         If $u < \exp\left[\frac{f(X_{proposal}) - f(X)}{T}\right]$ then $X := X_{proposal}$ and $c := c_{proposal}$, else
            $X := X$ and $c := c$;
•     Until iteration count = niter;
•     Decrease $T$ according to the temperature schedule;
•     Change template $t := X$;
•   Until stopping criterion = true;
```

- *X is the approximation to the optimal solution;*
- *End.*

□

According to (Kvasnička and Pospíchal 1995), the resulting *Messy Simulated Annealing* (MSA) algorithm is a robust and efficient stochastic optimisation method, which is able to correctly find the maxima of deceptive or highly multimodal objective functions.

3.2.4 SA Summary

In this section I have examined the SA algorithm, an approach which has the ability to reach global (or near global) optimal solutions, according to the hundreds of researchers that choose SA to solve their optimisation problems. Another characteristic of the approach is generality; for instance there are no restrictions regarding convexity and thus the method can handle objective functions with multiple local maxima. By also occasionally accepting downhill moves the algorithm manages to escape from local maxima, and to achieve results close to the global best. The algorithm is also simple to implement, without involving any unusual tricks and approaches, and is easy to program in any computer language. As far as the quality of solutions obtained by the algorithm, for many applications this is at least as good as, and sometimes much better than, those obtained by other algorithms, at least according to the literature.

However, the disappointingly long running times needed even to approximate convergence to the optimum, combined with the realisation that fine-tuning of the cooling schedule and a careful choice of neighbourhood structure are needed to get the best out of annealing, diminished some of the initial enthusiasm. The solution to all these problems is to start making modifications to the basic algorithm. But although these modifications improve the algorithm's performance, they also increase the number of decisions which must be made by the designer, and thus the technique loses its simplicity and robustness. So the decision goes to the designer.

On the one hand you can use the basic algorithm, with the most obvious neighbourhood structure, a geometric schedule, and a starting temperature determined by only a few experiments, which is easy to implement and will probably give you reasonable results; or on the other hand you can use a more complicated version of the algorithm, with a cooling schedule determined as a result of extensive experimentation, a neighbourhood structure decided by in-depth knowledge of the problem characteristics. This is difficult to implement but also may result in a powerful problem-specific approach. Of course you must not forget that even if you have spent months trying to find the best values of your parameters, you are always going to be uncertain about them and about the result of your algorithm. For example one of the major problems is knowing when to stop, as there is always the feeling that the next modification or parameter change may lead to dramatic improvements in solution quality or computational requirements.

Overall it seems that SA is generally an applicable, flexible, robust and easy-to-implement approximation algorithm, which is able to obtain near–optimal solutions for a wide range of optimisation problems. However, computation times can be long and in a number of cases other algorithms can be executed far quicker. For problem areas where other algorithms are not practical, SA appears to be a powerful optimisation tool.

3.3 Threshold acceptance (TA)

Threshold acceptance (TA) is a stochastic optimisation algorithm proposed by (Dueck and Scheuer 1990) whose structure is similar to—perhaps even simpler than—SA. Because of the similarity of the two algorithms, I will not give full details about implementation and modification problems; see Section 3.2 for more information.

Suppose again that our aim is to find the maximum of the objective function $f\colon S \longrightarrow \Re$ over a finite set S. TA starts with an element $x \in S$, which might be randomly chosen. Then, a high number of iterations is performed. In each iteration step the algorithm tries to replace its current solution with a new one, which is randomly chosen from the neighbourhood $N(x)$ of the present solution. Suppose that the new candidate is y. Then in SA y is always accepted if $f(y) \geq f(x)$ and with a positive probability (proportional to the current temperature) if $f(y) < f(x)$. On the other hand in TA y is accepted if and only if

$$f(x) - f(y) < T, \tag{3.11}$$

for a given positive *threshold value T*. So the essential difference between SA and TA consists of the different acceptance rules. TA accepts every new configuration which *is not much worse* than the old one, while SA accepts worse solutions only with rather small probabilities.

By allowing worse moves to be accepted during TA, it becomes possible to escape local maxima with regard to the given neighbourhoods. Usually at the beginning of the algorithm the threshold value has a large positive value, so almost all moves are accepted, while eventually the threshold is dropped to a positive value close to zero, and so only improving solutions are accepted.

So we must have a sequence of positive decreasing numbers to use as our threshold sequence during the algorithm. In (Dueck and Scheuer 1990), the threshold sequence is exogenously given. Another idea is given by (Winker and Fank 1997), who generate their threshold sequence from the empirical distribution of discrepancies $|f(x) - f(y)|$ as the run progresses. Finally a geometric sequence can be used (Volker and Henrik 1995). If T_0 is the starting value of the sequence, and T_f the final one, then

$$T_{new} = T_{old}(1 - \epsilon), \quad \text{where} \quad \epsilon = 1 - \left(\frac{T_f}{T_0}\right)^{\frac{1}{M}}, \tag{3.12}$$

with M the overall desired number of iterations. According to (Chipman and Winker 1995), the choice of the threshold parameters is not too crucial for the mean performance of the algorithm as long as it falls in a reasonable range.

An apparent advantage of TA is its greater simplicity. It is not necessary to compute probabilities or to make decisions. Also according to (Dueck and Scheuer 1990), TA yields better results than SA (possibly in a considerably smaller amount of time or "new configuration choice steps"). The algorithm can be stated as follows:

Algorithm 3.4. THRESHOLD ACCEPTANCE (TA):

```
• Begin.
•     Choose a configuration $i_{start}$;
•     Select the initial and final threshold values $T_0, T_f > 0$;
•     Select the threshold sequence schedule;
•     $i := i_{start}$ and $T := T_0$;
•     Repeat:
•       Repeat:
•       Generate a new configuration $j$ from the neighbourhood of $i$;
•       If $f(i) - f(j) < T$ then $i := j$ else $i := i$;
•       Until iteration count = niter;
•       Decrease $T$ according to the threshold sequence schedule;
•     Until stopping criterion = true;
•     $i$ is the approximation to the optimal solution;
• End.
```

□

3.4 Genetic algorithms (GA)

The *genetic algorithm* (GA) was first introduced by (Holland 1975), and has become a popular method for solving large optimisation problems with multiple local optima. Many researchers have since claimed success with GA in a broad spectrum of applications.

To call the GA paradigm "modern" today might seem to be stretching the truth, since it was first developed 30 years ago. Holland and his associates at the University of Michigan began to develop it in the 1960s and 1970s, but it was Holland's 1975 book, *Adaptation in Natural and Artificial Systems*, where the first full, systematic and theoretical treatment of GA was contained. After that we have a large range of books and articles on GA with many successful applications, ideas for improvements, and results. For instance, (Goldberg 1989) and (Davis 1991) both provide a useful description of the algorithm and a number of applications in a range of problems. Also interesting applications of the algorithm can be found in recent articles, including (Michalewicz and Janikow 1991), (South et al. 1993), (Rawlins 1991), (Whitley 1992), and (Franconi and Jennison 1997), where GA is compared with SA for a statistical image reconstruction problem.

3.4.1 Biological terminology

The name GA originates from the analogy between the representation of a complex structure by means of a vector of components, and the idea, familiar to biologists, of the genetic structure of a *chromosome.* In this subsection I will introduce some of the biological terminology that will appear throughout this section.

All living organisms consist of cells, and each cell contains the same set of one or more strings of DNA called chromosomes. The chromosomes can also be divided into *genes*, functional blocks of DNA, each of which encodes a particular protein. For instance, a particular gene can represent the eye colour of the organism. Then the different settings that this eye colour can take (e.g., brown, blue, etc.), are called *alleles.* The position of each gene on the chromosome is called the *locus.* The organisms may have multiple chromosomes in each cell. The complete collection of genetic material is called the organism's *genome.* Two individuals that have identical genomes are said to have the same *genotype.* The genotype gives rise, under fetal and later development, to the organism's *phenotype*–its physical and mental characteristics. Finally, organisms can be *diploid*, when their chromosomes are arrayed in pairs, or can be *haploid* otherwise. In order now to produce a new offspring, a *crossover* operation occurs. In each parent, genes are exchanged between each pair of chromosomes to form a *gamete* (a single chromosome), and then gametes from the two parents pair up to create a full set of chromosomes. Off-spring also are subject to the *mutation* operator, in which single *nucleotides* (elementary bits of DNA) are changed from parent to off-spring. The *fitness* of the organism, finally, is defined as the probability that the organism will live to reproduce, or as a function of the number of off-spring the organism has.

In GA the term chromosome typically refers to a candidate solution to a problem, and most often is simply a binary 0–1 string. The genes are either single bits or short blocks of adjacent bits that encode a particular element of the candidate solution. The allele is either 0 or 1. The crossover operation is simply an exchange of sections of the two parents' chromosomes, while mutation is a random modification of the chromosome which can be done by flipping the bit at a randomly chosen locus.

3.4.2 The algorithm

Suppose that our goal is to maximise a function $g(X)$ of the vector $X = (X_1, X_2, \ldots, X_p)$, where each $X_i, i = 1, 2, \ldots, p$, is binary (taking the value 0 or 1). In the case that your vector is continuous it is usual to replace the variables by binary expansions in order to run the algorithm. The basic GA starts by randomly generating an even number n of binary strings of length p to form an initial population,

$$\begin{array}{c} X_1^1, X_2^1, \ldots, X_p^1 \\ X_1^2, X_2^2, \ldots, X_p^2 \\ \vdots \quad \vdots \quad \vdots \\ X_1^n, X_2^n, \ldots, X_p^n \end{array} \tag{3.13}$$

A positive fitness f then is calculated as a monotone increasing function of g for each string in the current generation and n *parents* for the next generation are selected,

$$\begin{array}{c} Y_1^1, Y_2^1, \ldots, Y_p^1 \\ Y_1^2, Y_2^2, \ldots, Y_p^2 \\ \vdots \quad \vdots \quad \vdots \\ Y_1^n, Y_2^n, \ldots, Y_p^n \end{array} \tag{3.14}$$

with replacement, with the probability p_j of choosing the j^{th} string in the current population proportional to its fitness f_j, i.e.,

$$p_j = \frac{f_j}{\sum_{i=1}^{n} f_i}. \tag{3.15}$$

The new parents are considered in pairs; for each pair we perform a crossover operation with a pre-selected probability p_c. If crossover occurs, an integer k is generated from the uniform distribution $[1, p-1]$ and the last $(p-k)$ elements of each parent are exchanged to create two new strings. So, e.g., for the first pair of parents, suppose that crossover occurred, so we create two new strings,

$$\begin{gathered} Y_1^1, Y_2^1, \ldots, Y_k^1, Y_{k+1}^2, Y_{k+2}^2, \ldots, Y_p^2 \\ Y_1^2, Y_2^2, \ldots, Y_k^2, Y_{k+1}^1, Y_{k+2}^1, \ldots, Y_p^1 \end{gathered} \tag{3.16}$$

If crossover does not occur, the parents are copied unaltered into two new strings. After the crossover operation, we perform a mutation operation with a pre-selected probability p_m. If mutation occurs, we simply switch the value at each string position from 0 to 1 or vice versa. Mutation occurs independently at each element of each string.

The algorithm is allowed to continue for a certain number of generations. On termination, the string in the final population with the highest value of g can be returned as the solution to our optimisation problem. But, since a good solution may be lost during the algorithm, a more efficient strategy is to note the best, or even better the $\gamma\%$ best, solutions seen at any stage (for some reasonable γ) and return these as a solution.

The population size n, parameters p_c and p_m and fitness function f must be specified before the algorithm is applied. It is often reasonable to take f equal to g, but in some problems a more careful choice may be required. In the next subsections I discuss parameter choice and general implementation aspects of the algorithm. The algorithm can be stated as follows:

Algorithm 3.5. GENETIC ALGORITHM (GA):

```
• Begir Begin;
•   Ge  Generate randomly an even number n of individuals $X^i$ of length p;
•   Ev  Evaluate the fitness f of each individual;
•   Re  Repeat:
•     S   Select n new individuals $Y^i$ by replacement with probability proportional
          to f;
•     F   For every pair of $Y^i$ do:
•       Generate random $r_1$ uniformly in the range (0,1);
•     If $r_1 \le p_c$ then generate a uniform random integer in the range $[1, p-1]$
        and exchange the $(p-k)$ elements of each parent $Y^i$;
•    For every individual $Y^i$ do:
•      For every bit in the individual do:
•        Generate random $r_2$ uniformly in the range (0,1);
•        If $r_2 \le p_m$ switch the current element from 0 to 1 or vice versa;
•    Save the best $a\%$ individuals for the next run, with $Y_{opt}$ the best;
•    Generate the remaining $n(1-a)$ individuals randomly and calculate their
       fitness $f$;
•  Until stopping criterion = true;
•  $Y_{opt}$ is the approximation to the optimal solution;
• End.
```

□

Again a number of decisions must be made in order to implement the above algorithm. The hope is that in simple problems GA will rapidly find optimal or near optimal solutions. There are two major sources of difficulty for the algorithm. Firstly, it has been noted that the proportion of the population having the optimal value at each string position does not necessarily increase all the way to 1 as the algorithm progresses and the likelihood of many such elements appearing together to form an optimal solution can be very low. The second problem is that of *genetic loss*, when all copies of the optimal value at a certain element of X disappear, despite the presence of multiple copies in the initial population. Both these problems appear to contradict the implications of the *Schema Theorem*, that a beneficial schema such as the value 1 at any element of X in our examples is likely to propagate throughout the population once a single copy occurs.

3.4.3 The Schema Theorem

The word *schema* comes from the past tense of the Greek word $\epsilon\chi\omega$ (echo, to have), whence it came to mean shape or form. Suppose that we have two chromosomes

$$\begin{array}{c} 1\,1\,0\,1\,1\,1\,0 \\ 1\,0\,0\,0\,1\,1\,1 \end{array} \tag{3.17}$$

Then both are example of the schema

$$1 * 0 * * * *, \tag{3.18}$$

where the symbol * denotes string positions at which no value is stipulated and thus it can be replaced by a 0 or a 1. So the above schema $H = (1 * 0 * * * *)$ specifies $X_1 = 1, X_3 = 0$ in a vector X of length 7. It is clear that the chromosomes are also instances of several other schemata. Some schemata may contain both, while some others contain only one. It is obvious that each chromosome is an instance of 2^p distinct schemata, where p is the string's length. So, each time we evaluate the fitness of a chromosome we can say that we are gathering the schemata of which it is an instance. In a population of size n, we have $n\,2^p$ schemata, but of course in practice these numbers may be different, since we might have some overlappings or have no representatives at all. Under certain plausible assumptions, it can be shown that processing a population of size n in one generation processes $O\left(n^3\right)$ schemata. It is clearly impossible to store the average fitness value explicitly. A solution to this problem has been given by Holland through his well-known *Schema Theorem.*

Before we state some lemmas and the theorem, here is some notation that is used below. We call the *length* of a schema the distance between the first and the last defined (i.e., non *) position on the schema, and order the number of defined positions. Thus the previously described schema $H = (1 * 0 * * * *)$ has length 2 and order 2. The *fitness ratio* is the ratio of the average fitness of a given schema to the average fitness of the population. I am now ready to give the theorem together with some useful lemmas; see (Holland 1975) for proofs and further details.

Lemma 3.1. Under a reproductive plan in which a parent is selected in proportion to its fitness, the expected number of instances of a schema S at time $(t+1)$ is given by

$$E(S, t+1) = f(S, t)N(S, t), \tag{3.19}$$

where $f(S,t)$ is the fitness ratio for the schema S, and $N(S,t)$ is the number of instances of S, at time t.

Lemma 3.2. If crossover is applied at time t with probability p_c to a schema S of length $l(S)$, then the probability that S will be represented in the population at time $(t+1)$ is bounded below by

$$P(S, t+1) \geq 1 - \frac{p_c\, l(S)}{p-1}[1 - P(S,t)], \tag{3.20}$$

where p is the length of the chromosome.

Lemma 3.3. If mutation is applied at time t with probability p_m to a schema S of order $k(S)$, then the probability that S will be represented in the population at time $(t+1)$ is bounded below by

$$P(S,t+1) \geq 1 - p_m k(S). \tag{3.21}$$

Combining these results together we obtain the following theorem.

Theorem. SCHEMA: *Using a reproductive plan in which the probabilities of crossover and mutation are p_c and p_m, respectively, and schema S of order $k(S)$ and length $l(S)$ has fitness ratio $f(S,t)$ at time t, then the expected number of representatives of schema S at time $(t+1)$ is bounded below by*

$$E(S,t+1) \geq \left\{1 - \frac{p_c l(S)}{p-1}[1 - P(S,t)] - p_m k(S)\right\} f(S,t)N(S,t). \tag{3.22}$$

It follows from this theorem that short low-order schemata will increase their representation provided their fitness ratio is slightly more than 1. So the ideal situations are those where short, low-order schemata combine with each other to form better and better solutions.

According to (Jennison and Sheehan 1995) we cannot easily create a similar lower bound for the expected number of representatives of schema S at time $(t+k)$, where $k \geq 2$, and that provides a limitation of Holland's result. These authors also noted that when the fitness ratio falls below 1 copies of the schema will start to disappear, and then we would face the problem of genetic loss.

3.4.4 Implementation of GA

As was the case with SA, the algorithm implementer has to take a large number of decisions in order to run the algorithm.

First of all: how many times should you run the algorithm? The best number of runs, or of generations, would be one large enough to find the optimal solution, but also small enough to reduce as much as possible the computation time. A crucial question here is in each repetition of the algorithm how many models we randomly generate. One approach is at the end of each repetition to clear the current population and to produce randomly, at the beginning of the new repetition, a new initial population. By doing this we are forcing the algorithm to look into new regions, but on the other hand if the reproduction of the new population is computationally expensive we lose time. The other extreme is to save time and move the last population of the last repetition to the new repetition as it is; a compromise is to keep a specific percentage a% of models from the current population in the new repetition and randomly generate the rest.

Another decision has to do with the population size n, a number that of course depends on the length of each string. Some researchers use small population sizes in order to be able to repeat the algorithm longer, but others prefer to repeat the algorithm fewer times in order to use large population sizes. Which of the two methods is better? It is not clear. But because you will probably have a limit to your computation time, you will have to find a tradeoff between the population and generation sizes, since it would be impossible to use large numbers for both of them. The best thing that you can do is to run your algorithm many times, by using different population and generation sizes each time, and by trying a very large population and small generation size, and also a very large generation and small population size, you can discover which choice is best suited to your problem.

Implementation of GA also requires two probabilities, the probability of crossover p_c and the probability of mutation p_m. Here things are clearer. Almost all researchers agree that the probability of crossover must be fairly high (above 0.3), while the probability of mutation must be quite small (less than 0.1). Many researchers have spent a lot of time trying to find the best values of these parameters. (De Jong 1975), for instance, ran a lot of experiments and at the end he indicated that the best population size was 50–100 individuals, the best single-point crossover rate was 0.6 per pair of parents, and the best mutation rate was 0.001 per bit. These settings became widely used in the GA community, even though it was not clear how well GA would perform with these settings on problems outside De Jong's test suite. (Schaffer et al. 1989) spent over a year of CPU time systematically testing a wide range of parameter combinations. They found that the best settings for population size, crossover rate and mutation rate were independent of the problem in their test suite. Finally a lot of work has been done by (Grefenstette 1986). He suggested that in small populations (30, for example) it is better to use high values of the crossover rate, such as 0.88. The best crossover rate decreases to 0.50 for population size 50 and to 0.30 for population size 80. Finally in most of his runs he used a mutation rate of 0.01. The runs with mutation rate above 0.1 are more like a random search, but also the absence (or a very small value) of mutation is also associated with poorer performance.

Finally the implementer has to choose a good fitness function f. The theory says f must be positive and a monotone increasing function of your objective function g. Also some researchers prefer their fitness function to take values in the interval $[0, 1]$. The choice is wide; the only guide is that your function must provide sufficient selectivity to ensure that the algorithm prefers superior solutions to the extent that it eventually produces an optimal (or at least near-optimal) answer. On the other hand, selectivity must not be so strong that populations polarise at an early stage and the potential advantages of maintaining a diverse collection of solutions are lost. A frequent choice is $f = g$, probably the simplest one. Of course other more complicated choices are also possible and they might be sometimes advantageous. For example you can use $f = \exp(g)$, or $f = M + g$ for a suitable constant M. Finally if you want to use perfectly the definition and produce a positive, monotone increasing function of g which has values from 0 to 1, the best solution is probably to use

$$f(X) = \frac{g(X) - \min[g(X)]}{\max[g(X)] - \min[g(X)]}, \tag{3.23}$$

where $\max[g(X)]$ and $\min[g(X)]$ are (at least rough estimators of) the maximum and minimum values of g, respectively.

There appear to be no guaranteed general rules for implementing GA. From problem to problem there may be a huge difference, and so each problem needs its own attention. The best thing you can do is run GA many times, by using different settings each time, and at the end note which one performs best and use it from then on.

3.4.5 Modifications

In this section I cover the most important and effective modifications of GA which have proved useful for a number of different problems. Again, as in the SA case, it is probably better to consider these modifications only in situations where the simple algorithm fails to provide satisfactory results.

- **Population size**

We have already discussed the problem of population size in the previous section. A question that really bothers a lot of researchers is how the performance of GA is influenced by the population size. Obviously by using small populations we take the risk of under-covering the solution space with the unfortunate result of never finding a near-optimal solution, while on the other hand by using large populations we start having serious computational delays and our algorithm becomes very slow. (Goldberg 1989) reports that the optimal size for binary-coded strings grows exponentially with the length of the string p. Of course if this is true the practical performance of GA would be quite uncompetitive, especially in the case of large p, with other optimisation methods. But fortunately, there are many authors that agree that population sizes as small as 30 are often quite adequate. Also (Alander 1992) suggests that a value between p and $2p$ is not far from optimal for many problems.

- **Seeding**

What kind of population to use initially (how to *seed* GA) also bothers a lot of researchers. The most common idea is to randomly generate strings of 0s and 1s and then proceed with the algorithm. On the other hand there are others who believe that starting GA with a population of high-quality solutions, probably obtained from another heuristic technique, can help the algorithm to find a near-optimal solution quicker. It is also possible, however, that the chance of premature convergence to an optimum that is only local may be increased with this strategy.

- **Selection mechanisms**

In the original GA, parents are selected by means of a stochastic procedure from the population, and then a complete new population of offspring is generated which then replaces their parents. In one variation of this idea, each offspring could replace a randomly chosen member of the current population as it is generated. (De Jong 1975) introduced the idea of *generation gap*, where a proportion G was selected for reproduction, and their offspring replaced randomly selected existing population members.

Studies seem to indicate that GA performs better when populations do not overlap, but in the case of incremental replacement we have the advantage of preventing the occurrence of duplicates. So we are not wasting resources on evaluating the same fitness twice, and also we are not distorting the selection process, by giving more chances to a duplicate chromosome to reproduce.

Another good idea is to force the best member of the current population to be a member of the next as well. With this method we keep track of the best solution through the whole algorithm. Also (De Jong 1975) used an expected value model, where chromosomes are forced to become parents more or less in line with their expected frequencies as predicted by their fitness values, by following a policy of random sampling without replacement.

Another idea is to compare the parents with the offspring, and instead of copying the offspring directly to the new population, to copy the two best among the four (the two parents and the two children) to the new population. So if the parents for instance are "fitter" than their children, then they both survive, and we copy them to the new population. This is called an *elitist* strategy.

Finally, consider the problem of having two chromosomes which are close to different optima and the result of a crossover operator on them is worse than either. To address this (Goldberg and Richardson 1987) defined a sharing function, over the population which is used to modify the fitness of each chromosome, which can take a simple linear form:

$$h(d) = \left\{ \begin{array}{ll} 1 - \frac{d}{D} & \text{if} \;\; d < D \\ 0 & \text{if} \;\; d \geq D \end{array} \right\}, \tag{3.24}$$

where d is the distance between 2 chromosomes (an obvious measure could be Hamming distance, but this may cause problems; for more details refer to (Goldberg and Richardson 1987)) and D is a parameter. So for each pair of chromosomes we evaluate $h(d)$ and then we calculate

$$\sigma_j = \sum_{i \neq j} h(d_i) \tag{3.25}$$

for each chromosome j. Then we divide the fitness of each chromosome j by σ_j, and we replace the old fitness with the new values. The result of this will be that chromosomes which are close will have their fitness devalued in relation to those which are fairly isolated.

- <u>**Fitness calculation**</u>

I have already mentioned some things about the fitness function. Its selection is probably one of the most important and hardest decisions the implementer has to make. I have already noted the risk of using the objective function as a fitness and have given an example of a fitness function which seems to work fine in many cases. Another approach is to use a scaling procedure based on the transformation

$$f = \alpha g + \beta, \tag{3.26}$$

where g is the objective function and α and β are obtained so that f obeys the following conditions:

$$\begin{aligned} \text{mean}(f) &= \text{mean}(g) \\ \max(f) &= \mu - \max(g), \end{aligned} \tag{3.27}$$

where μ is a constant. Another idea is to ignore totally ·the objective function and use a ranking procedure instead. (Baker 1985) and (Reeves 1992) worked on these areas with apparent success. In this approach potential parents are selected with probability

$$P([k]) = \frac{2k}{n\,(n+1)}, \tag{3.28}$$

where $[k]$ is the k^{th} chromosome ranked in ascending order. So the best chromosome $[n]$ will be selected with probability $\frac{2}{n+1}$, roughly twice that of the median, whose chance of selection is $\frac{1}{n}$. (Whitley 1989) also worked with this method and he concluded that ranking should be preferred to scaling.

Finally an alternative is to use *tournament selection* (Goldberg and Deb 1991). Suppose we have n chromosomes, and successive groups of T chromosomes are taken and compared. We choose only the best one as a parent. When the n chromosomes are exhausted, another random permutation is generated. The whole procedure is repeated until n parents have been chosen this way. Each parent is then mated with another chosen purely at random. It is obvious that the best chromosome will be chosen T times in any series of n tournaments, the worst not at all, and the median on average once.

- **Crossover operators**

In most applications, the simple crossover operator described previously has proved extremely effective. However there are problems where more advance crossover operators have been found useful. First of all consider the *"string of chance"* crossover. Suppose we have the two chromosomes

$$\begin{array}{c} 1\,1\,0\,1\,0\,0\,0\,1 \\ 1\,1\,0\,1\,1\,1\,0\,0 \end{array} \tag{3.29}$$

These chromosomes have the same elements in the first four positions. Cases like this are quite common, especially in the later stages of the algorithm. If the crossover point is any of these four first positions, then the new string will not be a different chromosome. (Booker 1987) and (Fairley 1991) have both suggested that it is better to examine the parents and find the crossover points which would produce different offspring, before applying the crossover operator. It was Fairley who implemented this idea by using the "string of chance" crossover, which entails computing an *exclusive-or (XOR)* between the parents. Only positions between the outermost 1s of the XOR string will be considered as crossover points. In our example, the XOR string is

$$0\,0\,0\,0\,1\,1\,0\,1. \tag{3.30}$$

So only the last four positions will give rise to a different offspring.

In the simple crossover operator we randomly choose a single position and we exchange the elements of the two parents, but there is no reason why the choice of crossover points should be restricted to a single position. Many researchers have claimed large improvements with the use of *multi-point crossovers.* Consider the simplest case of *two–point crossover* and suppose we have the following strings:

$$\begin{array}{cc|cccc|cc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

If the chosen positions are the third and the seventh then we can produce the following offspring:

$$\begin{array}{c} 0\,1\,0\,0\,0\,1\,0\,0 \\ 1\,0\,1\,0\,1\,1\,1\,0 \end{array} \tag{3.31}$$

by taking the first two and the last two elements from the one parent and the rest from the other each time. Of course this is not the only kind of exchange we can make. In the above example, for instance, we can randomly choose again two positions in the one parent (suppose we again pick the third and seventh), and then we can copy the third and the seventh elements of this parent and the rest from the other. With this method we produce the following offspring:

$$\begin{array}{c} 1\,0\,0\,0\,1\,1\,0\,0 \\ 0\,1\,1\,0\,0\,1\,1\,0 \end{array} \tag{3.32}$$

The operator that has received the most attention in recent years is the *uniform crossover.* It was studied in some detail by (Ackley 1987) and popularised by (Syswerda 1989). Suppose we have the following strings:

$$\begin{array}{c} 1\,0\,0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,0\,0\,1\,1 \end{array} \tag{3.33}$$

Then for each position randomly (with probability usually 0.5) pick each bit from either of the two parent strings. If you want to produce two offspring you can do the above twice. So for example we can produce the following offspring:

$$\begin{gathered} 0\,0\,0\,1\,0\,0\,1\,0 \\ 1\,1\,0\,0\,1\,0\,1\,1 \end{gathered} \tag{3.34}$$

In the first offspring, we have chosen the second, third, sixth, seventh, and eighth element from the first parent, and the rest from the second, and in the second offspring we have chosen the first, third, fourth, fifth, sixth and seventh bit from the first parent, and the rest from the second.

A modified version of the uniform crossover is the version that the *CHC Adaptive Search Algorithm* (Eshelman 1991) uses, which I will call *highly uniform crossover.* This version crosses over half (or the nearest integer to $\frac{1}{2}$) of the non-matching alleles, where the bits to be exchanged are chosen at random without replacement. So for example if we have again the following parents,

$$\begin{gathered} 1\,0\,0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,0\,0\,1\,1 \end{gathered} \tag{3.35}$$

we can see that the non-matching alleles are the first, second, fourth, fifth, and eighth, which are five in total. So we are going to cross three of them randomly, (say) the first, second and fourth, and so the children produced will be

$$\begin{gathered} 0\,1\,0\,1\,1\,0\,1\,0 \\ 1\,0\,0\,0\,0\,0\,1\,1 \end{gathered} \tag{3.36}$$

With this operator we always guarantee that the offspring are the maximum Hamming distance from their two parents.

(Sirag and Weisser 1987) used a different kind of *generalised crossover*, by modifying the basic genetic operators in the spirit of SA. Thus, for example, the crossover operator is modified by a *threshold energy* θ_c which influences the way in which individual bits are chosen. Briefly, as the offspring chromosome is generated, there is a presumption in favour of taking bit $(i+1)$ from the same parent as bit i. However, bit $(i+1)$ is taken from the other parent with probability

$$\exp\left(\frac{-\theta_c}{T}\right), \tag{3.37}$$

where T is a "temperature" parameter, which is slowly decreasing according to an "annealing schedule". At high temperatures, this can be expected to behave rather like the generalised uniform operator; as temperature moderates the number of switches between parents decreases and it becomes more like the standard simple crossover, while at very low temperatures it just copies one of the parents. Choosing the best values of θ_c and T may require some experimentation, as will the annealing schedule.

These are the most common crossover operators. It is worth noting that there is no similar section about mutation operators. This is not because there are not other mutation operations available in the literature, but because I believe that for a binary-coding chromosome the best idea is to use the standard one. Also mutation is probably not as important as crossover in the problem posed in this dissertation.

- **Inversion**

In simple versions of GA the order of the elements in the string can have a large effect on performance, because entire blocks of elements are crossed over together; this will act, for example, to prevent interchanging the first element of one parent with the last element of the other. One possibility for solving this problem is uniform cross-over; another approach involves *inversion*, an operator which takes account of order relationships. You can use it together with the crossover operator, and so produce a larger variety of offspring. For instance suppose we have the following chromosome:

$$1\,0\,1\,1\,0\,0\,1\,0. \tag{3.38}$$

We randomly choose two positions with different elements (the second and seventh, say), and we just exchange these two positions (a *two-bit swap*). So the new chromosome will look like

$$1\,1\,1\,1\,0\,0\,0\,0. \tag{3.39}$$

- **Adaptive operator probabilities**

As we have already seen in simple GA, the probability of using either crossover or mutation is fixed throughout, and usually crossover is applied with a high probability (above 30–50%) while mutation is applied with low frequency (less than 1%). (Reeves 1992) found it useful to change this approach, and allow the mutation rate to change during the run. The idea was to make the mutation rate inversely proportional to the population diversity, in order to prevent or at least alleviate the problem of premature convergence. (Booker 1987), on the other hand, used an adaptive crossover rate, where the rate was varied according to a characteristic called *percent involvement*.

(Davis 1991) has further suggested that either crossover or mutation should be applied at each iteration but not both. So at each step the algorithm has to decide which operator is going to be used and this is achieved on a basis of a probability distribution which he calls *operator fitness*. Usually we choose a number like 75% for crossover fitness, and thus the crossover would be chosen three times as often as mutation. In further development of this idea, we can allow operator fitness to change during the run. A possibly good idea is to start with a high crossover fitness, since crossover is more important at the beginning when the population is diverse, but as the chromosomes start to converge it is clearly important to increase the chance of finding different solutions, which can then be done only with mutation.

- **Chromosome coding**

Although the 0–1 binary setting is the obvious one in which to use GA, and also is the one that we have to use for our problem, in this section I briefly describe problems that have inputs with real values. In this case we can simply map the real numbers onto binary strings of a desired degree of precision. Thus if an input takes values in the interval $[a, b]$ and is mapped to a string of length p, then the precision is

$$\frac{b-a}{2^p-1}. \tag{3.40}$$

The problem in this case is that values which are close in the original space may be far away in the new binary-mapped space. For example suppose that the optimum value of a problem that we try to maximise is the real number 32, and we are using a 6-bit chromosomes, so the binary optimal value is (1 0 0 0 0 0). A near-optimal solution is the real number 31, which maps to (0 1 1 1 1 1), a completely different string. Additionally the binary string (0 0 0 0 0 0), which is only one bit different from the optimum, represents the numerical value 0, which is far from the optimum. This led Caruna (Caruna and Schaffer 1988) to advocate the use of a Gray code mapping, but this introduces further problems in that there is no simple algorithm for decoding a Gray code.

Despite these problems there are several empirical comparisons between binary coding and real-valued or even multiple-character coding which have shown worse performance than the binary choice. Examples include the (Kitano 1990) many-character representation for graph-generation grammars, the (Meyer and Packard 1992) real-valued representation for condition sets, the (Montana and Davies 1989) real-valued representation for neural-network weights, and the (Schultz–Kremer 1992) real-valued representation for torsion angles in proteins.

So how is one to decide on the correct coding for one's problem? First of all you have to remember that the performance of your algorithm depends a lot on the problem and so there are no rigorous guidelines about which coding will work better. Probably a good thing to do is to follow the idea of (Davis 1991), to use whatever coding is the most natural on your problem and then devise a GA that can use that coding.

- **Sequence representation**

Many problems of interests, such as the famous *Travelling Salesman Problem*, can be most naturally represented as a *permutation*. The problem in this case is that the original crossover operator then does not work. We can see why through the following example. Suppose we have the following two parents:

$$\begin{array}{c} 1\,5\,2\,4\,3\,6 \\ 2\,5\,3\,6\,4\,1 \end{array} \tag{3.41}$$

Then if we assume that the crossover point is the fourth position we end up with the following offspring:

$$\begin{array}{c} 1\,5\,2\,6\,4\,1 \\ 2\,5\,3\,4\,3\,6 \end{array} \tag{3.42}$$

which of course are invalid.

Several researchers have tried to find solutions to this problem and define new operators. (Goldberg and Lingle 1985) defined the *partially mapped crossover.* This operator uses two crossover points, and the section between these points defines an interchange mapping. So in the above example suppose that the two crossover points are the second and fifth:

$$\begin{array}{c|ccc|cc} 1 & 5 & 2 & 4 & 3 & 6 \\ 2 & 5 & 3 & 6 & 4 & 1 \end{array}$$

These crossover points define the interchange mapping

$$\begin{aligned} 5 &\leftrightarrow 5 \\ 2 &\leftrightarrow 3 \\ 4 &\leftrightarrow 6 \end{aligned} \tag{3.43}$$

and so we end up with the following offspring:

$$\begin{matrix} 1\,5\,3\,6\,2\,4 \\ 3\,5\,2\,4\,6\,1 \end{matrix} \tag{3.44}$$

(Reeves 1992) used a *C1 operator* to solve a *flowshop sequencing problem.* This operator works by randomly choosing a point, taking the first bit from the one parent and filling the chromosome by taking in order each "legitimate" element from the other parent. In our example suppose that we randomly choose the third position,

$$\begin{array}{cc|cccc} 1 & 5 & 2 & 4 & 3 & 6 \\ 2 & 5 & 3 & 6 & 4 & 1 \end{array}$$

then our new offspring are going to be

$$\begin{matrix} 1\,5\,2\,3\,6\,4 \\ 2\,5\,1\,4\,3\,6 \end{matrix} \tag{3.45}$$

Finally, as was the case with binary inputs, with permutation problems another good idea is to use a uniform crossover. We generate randomly a *crossover template* of 0s and 1s, where 1s define elements taken from the one parent, while the other elements are copied from the other parent in the order in which they appear in that chromosome. Returning to the example, if we generate the template

$$1\,0\,1\,1\,0\,0 \tag{3.46}$$

then our two new offspring are

$$\begin{matrix} 1\,5\,2\,4\,3\,6 \\ 2\,1\,3\,6\,5\,4 \end{matrix} \tag{3.47}$$

Mutation also needs to be re-defined for the case of a sequence representation. (Reeves 1992) explored the *exchange mutation*, where you interchange two randomly chosen elements of the permutation (a two-bit swap). Another idea is the *shift mutation*, where you move a randomly chosen element a random number of places to the left or right. Finally (Davis 1991) used the idea of a *scramble sublist mutation*, in which you choose two points of the string at random, and randomly permute (scramble) the elements between these two positions.

- **Hybridisation**

If your GA does not perform well and you want to enhance its effectiveness, you can *hybridise* it with another heuristic in various ways. For example, many researchers have described ways in which local neighbourhood search or extensions such as SA can be embedded in GA in order to make it more effective.

(Goldberg 1989) described a method for incorporating neighbourhood search into a GA, in a procedure he calls *G-bit improvement.* What he simply did was to select periodically some of the best strings for a search of a neighbourhood defined by single-bit reversals. On completion of the neighbourhood search, the locally optimal strings were re-introduced into the population for the next phase of the algorithm. (Suh and Van Gucht 1987) also added a similar hybridisation to their GA, where heuristic operators based on two-optimal search and SA were used alongside simple crossover. Finally (Kapsalis et al. 1993) and (Beaty 1991) used a GA to make the "top-level" decisions on the form of a solution, which was then taken and solved by a problem-specific procedure. You can also devise special types of genetic operators, which are more powerful than the simple crossover and mutation.

I now examine two of the best-known hybrid GAs, *genetic local search* and *genetic simulated annealing.* The first of these has been proposed by several authors, mainly for solving the travelling salesman problem (Jog et al. 1989; Ulder et al. 1991). The algorithm has the following general form:

Algorithm 3.6. GENETIC LOCAL SEARCH:

- *Initialisation;*
- *(Local Search and termination test). Apply Local Search (Algorithm 3.1) to the n solutions in the current population. If a prespecified stopping condition is satisfied during the Local Search, stop the algorithm. If the Local Search is completed for all the n solutions, let the set of n obtained local optimal solutions be the current population, and go to the next step;*
- *(Selection);*

- *(Crossover);*

- *(Mutation);*

- *Keep the best* $100\,a\%$ *and randomly generate the remaining* $n(1-a)$ *individuals to create the new population;*

- *Return to the second step.*

□

The problem with this algorithm is the enormous computation time for finding n local optimal solutions in each generation. A way to decrease this computation time can be to search only a part of the neighbourhood solutions. For example, we can use the strategy to search a randomly selected 10% of the neighbourhood solutions in each local search procedure. If there are no solutions that improve the current one in the 10% neighbourhood solutions, the local search procedure is terminated without examining all the neighbourhood solutions.

Genetic SA has a similar general form:

Algorithm 3.7. GENETIC SIMULATED ANNEALING:

- *Initialisation;*

- *(SA and termination test). Apply SA (Algorithm 3.2) to the* n *solutions in the current population. If a prespecified stopping condition is satisfied during SA, stop the algorithm. If SA is completed for all the* n *solutions, let the set of* n *obtained local optimal solutions be the current population, and go to the next step;*

- *(Selection);*

- *(Crossover);*

- *(Mutation);*

- *Keep the best* $100\,a\%$ *and randomly generate the remaining* $n(1-a)$ *individuals to create the new population;*

- *Return to the second step.*

□

It appears to be better to apply SA with constant temperature to each of the n solutions in the current population, because by using constant temperature we manage to avoid extreme deterioration of the current solution during the initial state of annealing with high temperature. (Inshibuchi et al. 1995) suggested a way to modify SA, by randomly selecting k neighbourhood solutions of the current one and letting the best one be the candidate solution for the next transition in SA, in order to improve the performance of genetic SA.

- **Parallel implementation**

A serious drawback of GA is its inefficiency when implemented on a sequential machine. However, due to its inherent parallel properties, it can be successfully implemented on parallel machines, in some cases resulting in a considerable speedup. The first approach is to evaluate the fitness of each chromosome of the population in parallel. This approach has been used by several authors, including (Talbi and Bessière 1991) and (Mühlenbein et al. 1988). The second approach is to allocate sub-populations of chromosomes to parallel processors which proceed independently for a certain number of generations. You can simply insert the best chromosome that you have found amongst all sub-populations into each of them. (Pettey et al. 1987) took this route by donating and receiving the best individual once in every generation, while (Cohoon et al. 1987) preferred to copy a randomly chosen subset of solutions between the sub-populations following a relatively large number of generations.

3.4.6 Genitor algorithm

The *genitor algorithm* (Whitley 1989) was the first of what (Syswerda 1989) has termed the "steady state" genetic algorithms. There are three differences between the genitor algorithm and the "vanilla" (plain and simple) version of GA. First of all at the reproduction stage we only produce one offspring. Then this offspring is immediately placed back in the population, by replacing the least fit member. So at the end our population will consist of half of the parents (the "fittest" ones) and all the offspring. Finally the last difference is that in the genitor algorithm the fitness is assigned according to rank (Baker 1985; Whitley 1989), and by that we maintain a more constant selective pressure over the course of the search.

3.4.7 CHC adaptive search algorithm

The *CHC adaptive search algorithm* was developed by (Eshelman 1991). CHC stands for *cross-generational elitist selection*, *heterogeneous recombination* and *cataclysmic* *mutation.* This algorithm uses a modified version of uniform crossover, called *HUX*, where exactly half of the different bits of the two parents are swapped. Then if our population size is n, we draw the best n unique individuals from the parent and offspring populations to create the next generation. *HUX* is the only operator used by CHC adaptive search; there is no mutation.

In CHC adaptive search two parents are allowed to mate only if they are a certain Hamming distance (say d) away from each other. This form of "incest prevention" is designed to promote diversity. Usually we start with $d = \frac{p}{4}$, where p is the length of the string. If the new population is exactly the same as the previous one, we decrease d and we rerun the algorithm. When d becomes negative the result is the ***divergence procedure*** in which we replace the current population with n copies of the best member of the previous population, and for all but one member of the current population we flip $r \times p$ bits at random where r is the divergence rate (for instance the compromise value 0.5). We replace d by $d = r(1-r)p$ and restart the algorithm.

3.4.8 GA summary

GA is an algorithm which has become quite popular in recent years, as an approach to solving large and difficult optimisation problems. It is easy to write one general GA computer program to address many different problems. Also it has the advantage that it does not rely on unrealistic assumptions—such as linearity, convexity, or differentiability of the criterion function—in contrast with some other optimisation techniques. The only requirement is the ability to calculate a measure of performance, which may be highly complicated and non-linear. It is therefore evident that GA is quite robust. Furthermore, although it is possible to fine-tune GA to work better on a given problem, it is nonetheless true that a wide range of parameter settings (such as population sizes and crossover and mutation rates) will give acceptable results. Another advantage of the algorithm is the ability of the implementer to change it easily to model variations of the original problem, in contrast with other methods where even relatively minor modifications to the problem may cause severe difficulties. Finally there is great scope for implementing GA in parallel.

Unfortunately, however, despite these important features, it seems from the literature that GA may often be out-performed or fail badly even on simple problems. In many papers, for instance (Franconi and Jennison 1997), authors have tried to make comparisons between GA and other optimisation techniques, such as SA, and in most cases GA did not win. Also the algorithm might need disappointingly long running times to reach a near-optimal solution, especially in cases with high dimensions. It remains to be seen how it will do on the problem posed in this dissertation.

3.5 Tabu search (TS)

Tabu search (TS) is a "higher-level" heuristic procedure for solving optimisation problems, designed to guide other methods to escape the trap of local optima. Originally proposed by (Glover 1977) as an optimisation tool applicable to nonlinear covering problems, its present form was proposed 9 years later by the same author (Glover 1986), and with even more details several years later again in (Glover 1989). The basic ideas of TS have also been sketched by (Hansen 1986). Together with SA and GA, TS has been singled out by the *Committee on the next decade of Operations Research* in 1988 as "extremely promising" for future treatment of practical applications. This stochastic optimisation method does not appear to be well-known to statisticians (Fouskakis and Draper 1999), so I will spend more time than usual on reviewing its literature.

The two key papers on TS are probably (Glover 1989; Glover 1990a); the first analytically describes the basic ideas and concerns of the algorithm and the second covers more advanced considerations. (Glover 1990b), a tutorial, and (Glover et al. 1993), a users guide to TS, are also useful. Other authors who have made contributions include (Cvijovic and Klinowski 1995), who specialised the algorithm for solving the multiple minima problem for continuous functions, and (Reeves 1995), who devoted a whole chapter to TS in his book *Modern Heuristic Techniques for Combinatorial Problems.* In a variety of problem settings, TS has found solutions superior to the best previously obtained by alternative methods. A partial list of TS applications is as follows:

- Employee scheduling (Glover and McMillan 1986);
- Maximum satisfiability problems (Hansen and Jaumard 1990);
- Telecommunications path assignment (Oliveira and Stroud 1989);
- Probabilistic logic problems (Jaumard et al. 1991);
- Neural network pattern recognition (de Werra and Hertz 1989);
- Machine scheduling (Laguna et al. 1991);
- Quadratic assignment problems (Skorin–Kapov 1990);
- Travelling salesman problems (Malek et al. 1989);
- Graph colouring (Hertz and de Werra 1987);
- Flow shop (Taillard 1990);

- Job shop with tooling constraints (Widmer 1991);
- Just-in-time scheduling (Laguna and Gonzalez-Velarde 1991);
- Electronic circuit design (Bland and Dawson 1991); and
- Nonconvex optimization problems (Beyer and Ogier 1991).

3.5.1 The algorithm

Webster's dictionary defines *tabu* or *taboo* as "set apart as charged with a dangerous supernatural power and forbidden to profane use or contact ..." or "banned on grounds of morality or taste or as constituting a risk ...". TS scarcely involves reference to supernatural or moral considerations, but instead is concerned with imposing restrictions to guide a search process to negotiate otherwise difficult regions. These restrictions operate in several forms, both by direct exclusion of certain search alternatives classed as "forbidden" and also by translation into modified evaluations and probabilities of selection.

Suppose we want to maximise an objective function $f(X)$ of the vector X. TS is divided into three parts: *preliminary search*, *intensification*, and *diversification*. Preliminary search, the most important and the main part of the algorithm, works as follows. TS starts from an initial solution. Then amongst all neighbouring solutions, TS seeks the one with the highest value. This move might not lead to a better solution, but enables the algorithm to continue the search without becoming confounded by the absence of improving moves and to climb out of local optima. This is one of the characteristic properties of the algorithm. In TS we keep moving even if that means that we are going to a worse move. If there are no improving moves (indicating a kind of local optimum), TS chooses the one that least degrades the objective function. In order to avoid returning to the local optimum just visited, the reverse move now must be forbidden. This is done by storing this move, or more precisely a characterisation of this move, in a data structure—the *tabu list*—often managed like a circular list, empty at the beginning and with a first-in-first-out mechanism, so that the latest forbidden move replaces the oldest one. This list contains a number s of elements defining forbidden (tabu) moves; the parameter s is called the *tabu list size*. However, the tabu list may forbid certain relevant or interesting moves, as exemplified by those that lead to a better solution than the best one found so far. Consequently, an *aspiration criterion* is introduced to allow tabu moves to be chosen anyway if they are judged to be sufficiently interesting.

Suppose for illustration that we have to maximise a function of a binary vector of length 5, and take $(0, 1, 0, 0, 1)$ with criterion value 10 as the initial solution. Then if we define the neighbours to be just the vectors that we can create by changing each time each value from zero to one or from one to zero (*one-bit flips*), we obtain the following solutions $(X_1, \ldots, X_5)$, with hypothetical values of the criterion function:

Vector	$(1,1,0,0,1)$	$(0,0,0,0,1)$	$(0,1,1,0,1)$	$(0,1,0,1,1)$	$(0,1,0,0,0)$
Value	9	8	5.4	7.1	7.3

From the above possible moves, none leads to a better solution. But in TS we keep moving anyway, so we accept the best move among the five in the neighbourhood, which is the vector $(1,1,0,0,1)$ with value 9. This move becomes our new one. In order now to avoid going back to the previous solution we set tabu the move that changes X_1 from 1 back to zero. So among the next five neighbours,

Vector	$(0,1,0,0,1)$	$(1,0,0,0,1)$	$(1,1,1,0,1)$	$(1,1,0,1,1)$	$(1,1,0,0,0)$
Value	10	8.1	9.7	7.9	6.9

the first one is tabu, the rest non-tabu. The aspiration criterion is simply a comparison between the value of the tabu move and the aspiration value, which is usually the highest value found so far (in our example 10). So because our tabu move has value not larger than the aspiration value, it remains tabu, and so we have to choose among the other four. From these the one with the best solution is the third neighbour, $(1,1,1,0,1)$, with value 9.7. So now the move that changes X_3 from 1 to 0 is tabu as well. Suppose that the tabu list size has been set to 4 for this example, and continue the algorithm. Our next neighbours are

Vector	$(0,1,1,0,1)$	$(1,0,1,0,1)$	$(1,1,0,0,1)$	$(1,1,1,1,1)$	$(1,1,1,0,0)$
Value	5.4	9.2	9	3	6.5

The first and the third neighbours now are tabu, with values less than the aspiration value for both, and so we have to search among the other three. Between these three moves the best one is the second neighbour, $(1,0,1,0,1)$, with value 9.2. So now the move that changes X_2 from 0 to 1 is tabu as well. Going one more step produces the next neighbourhood,

Vector	$(0,0,1,0,1)$	$(1,1,1,0,1)$	$(1,0,0,0,1)$	$(1,0,1,1,1)$	$(1,0,1,0,0)$
Value	10.8	9.7	8.1	7.1	6.1

The first, second, and third moves according to the tabu list are tabu. But the first move has value larger that the aspiration value, and so its tabu status is cancelled. So the non-tabu moves now are the first, fourth, and fifth, and among these the best one is the first one, $(0,0,1,0,1)$, with value 10.8, which also replaces the best one found so far.

We continue doing the above loop for a specified number of iterations. Of course even for this part, the algorithm designer has to make several decisions, such as the initial solution, the tabu list size, the aspiration criterion, the neighbourhood structure and size (in case the size of the neighbourhood is very big it is probably better to search among a subset of the neighbourhood), and the stopping rule.

After finishing with the preliminary search, intensification starts. We go to the best solution found so far and clear the tabu list. The algorithm then proceeds as in the preliminary search phase. If a better solution is found, intensification is restarted. We can have a specified number of restarts; after that number the algorithm goes to the next step. Also if the current intensification phase does not find a better solution after a specified number of iterations, the algorithm goes to the next step. Intensification provides a simple way to focus the search around the current best solution. The designer here has to decide on the maximum number of restarts that the algorithm will be allowed, and how many iterations without improvement the algorithm will be allowed before going to the next part.

Finally we have the last part of the algorithm, diversification. We again clear the tabu list, and set the s most frequent moves of the run so far to be tabu, where s is the tabu list size. Then we choose a random state and proceed through the preliminary search phase for a specified number of iterations. Diversification provides a simple way to explore regions that have not been visited much. The designer here has to decide on the number of iterations that will be spent in this part.

After the end of the third part, either you report your best solution as the optimal result found, or (even better) you repeat the whole algorithm (all three parts). You can repeat the whole algorithm many times, subject to a pre-specified number. I will discuss parameter choice and general implementation aspects of the algorithm in the subsections below.

The algorithm more formally can be stated as follows:

Algorithm 3.8. TABU SEARCH (TS):

```
• Begin;
•     Randomly choose a configuration $i_{start}$ and set $i := i_{start}$;
•     Evaluate the criterion function $f(i)$;
•     Set the aspiration value $\alpha := lo$, a small number;
•     Determine $s := Listlength$, the length of the tabu list;
•     Set $Move := 0$ and $i_{max} := i_{start}$;
•     Repeat:
•         $\boxed{\text{Preliminary Search}}$
•         Add $i$ to the tabu list at position $s$;
•   Set $s := s - 1$. If $s = 0$ then set $s :=$ Listlength;
•   Set Move := Move + 1, $i_{nbhd} := i$, and $f_{nbhd} :=$ low, a small number;
•   For each neighbour $j$ of $i$ do:
•     If $f(j) > \alpha$ do:
•       If $f(j) \geq f_{nbhd}$ then set $i_{nbhd} := j$ and $f_{nbhd} := f(j)$;
•     If $f(j) \leq \alpha$ do;
•       If $j$ is in the tabu list go to the next neighbour;
•       Else if $j$ is non-tabu and $f(j) \geq f_{nbhd}$ then set $i_{nbhd} := j$ and
          $f_{nbhd} := f(j)$;
•   Set $\alpha := \max(\alpha, f_{nbhd})$ and $i := i_{nbhd}$;
•   If $f(i) \geq f(i_{max})$ then $i_{max} := i$;
•   If Move $\neq$ maxmoves go back to Preliminary Search;
•   Else go to Intensification;
•   Intensification
•   Repeat:
•     Set $i := i_{max}$ and clear the tabu list;
•     Repeat:
•       Do the Preliminary Search;
•     Until you find a better solution than $i_{max}$. If no improvements after $n_{int}$
        iterations go to the next part;
•   Until $n_{impr}$ runs;
•   Diversification
•   Clear the tabu list and set the s most frequent moves to be tabu;
•   Randomly choose a configuration $i$;
•   Evaluate the criterion function $f(i)$;
•   Repeat:
•     Do the Preliminary Search;
•   Until you have run it $n_{div}$ times;
•  Until you have run the whole algorithm rep times;
•  $i_{max}$ is the approximation to the optimal solution;
• End.
```

□

3.5.2 Implementation and modifications of TS

From the above algorithm, it is clear that the designer has to make decisions for a number of crucial elements of TS, such as neighbourhood sizes, types of moves, tabu list structures, and aspirations conditions. Also you have to define values for several parameters, such as $maxmoves$, n_{int}, n_{impr}, n_{div} and rep. There appears to be surprisingly little advice in the literature about how to make these choices. I will address this issue in Chapters 4 and 5 through simulation studies.

- **Neighbourhood sizes and candidate lists**

When the dimension of the problem is high, a complete neighbourhood examination may be expensive in terms of CPU-time. For this reason, it is probably better, in cases like this, to examine only the regions of the neighbourhood that contain moves with desirable features.

One way of doing this is to use a *neighbourhood decomposition strategy.* You simply decompose the neighbourhood into coordinated subsets at each iteration. A TS aspiration threshold, or other means of linking the examination of subsets, is commonly applied to limit the frequency of selecting moves from subsets whose current alternatives are gauged less attractive. This strategy may be able both to speed up the algorithm and to generate some diversity in the search. (Laguna et al. 1991) have successfully applied this idea, and managed to limit the domains of jobs that are shifted in machine scheduling, while (Fiechter 1994) has also used this idea in travelling salesman problems. Finally (Semet and Taillard 1993) have succeeded in cutting down computation times by a factor of three while simultaneously obtaining better solutions, by applying such a decomposition approach to cyclically scan about one fourth of all possible moves at each iteration.

Another technique is *elite evaluation candidate lists*, in which you store a collection of elite (highest evaluation, most promising) moves on a candidate list. Then at each iteration, the moves belonging to the candidate list are examined first, followed by a subset of the regular neighbourhood, gradually replacing candidate list moves that are no longer attractive. Periodically, after many iterations or when the quality of moves on the candidate list deteriorates below a chosen threshold, a significantly larger portion of the current neighbourhood is examined in order to reconstruct the candidate list. This technique is motivated by the assumption that a good move, if not performed at the present iteration, will still be a good move for some number of iterations. A simplified variant of this strategy is to perform every move on the elite candidate list in succession, provided the move remains valid when its turn arrives. (Glover et al. 1986) improved the technique by introducing an aspiration level threshold that moves must satisfy in order to be selected.

In some applications it can be advantageous to isolate certain attributes of moves that are expected also to be attributes of good solutions, and to limit consideration to those moves whose composition includes some portion of these "preferred" attributes. The *preferred attribute candidate list* technique seeks to organise moves so that they do not have to be composed entirely of some special elements. One or more of these elements are required to be incorporated in a "key segment" of a move, so that all moves containing such a segment can be generated efficiently.

Finally a type of candidate list that is highly exploitable by parallel processing is a *sequential fan candidate list.* The main idea is to generate the γ best alternative moves at a given step, and then to create a fan of solution streams, one for each alternative. The best few available moves for each stream are again examined, and only the γ best moves overall provide the γ new streams at the next step.

- **Attributes**

Suppose that our current solution is X_{now} and we want to create a tentative solution X_{trial}, with X a binary vector. The most common and natural types of attributes for a move X_{now} to X_{trial} are as follows:

1. Change of a selected variable x_i from 0 to 1.
2. Change of a selected variable x_j from 1 to 0.
3. The combined change of the previous two taken together.
4. Change of $f(X_{now})$ to $f(X_{trial})$.
5. Change of a function $g(X_{now})$ to $g(X_{trial})$.
6. Change represented by the difference value $g(X_{trial}) - g(X_{now})$.
7. The combined changes of the previous two types for more than one function g considered simultaneously.

Attributes 5–7 are based on a function g that may be strategically chosen to be completely independent from the objective function f. For example we can choose g to be a measure of the distance between the given solution and the best solution found so far. Then with attribute 6 we can see if the trial solution leads the search farther from or closer to the best solution found so far.

- <u>Tabu restrictions</u>

Suppose that a move that contains an attribute e from X_{now} to X_{next} is performed. Then the reverse attribute $\bar{e}$ has to remain tabu for some time in order to prevent reversing. Examples of kinds of tabu restrictions frequently employed are as follows. A move is tabu if

- x_j changes from 1 to 0 where x_j previously had changed from 0 to 1.
- x_i changes from 0 to 1 where x_i previously had changed from 1 to 0.
- At least one of the previous two occur. This condition is more restrictive than either the first or the second separately, as it makes more moves tabu.
- Both the first two restrictions occur. This condition is less restrictive than either the first or the second separately, as it makes fewer moves tabu.
- Both the first two restrictions occur, and in addition the reverse of these moves occurred simultaneously on the same iteration in the past.
- $g(X)$ receives a value u' that it received on a previous iteration, i.e., $u' = g(X')$ for some previously visited solution X'.
- $g(X)$ changes from u'' to u', where $g(X)$ changed from u' to u'' on a previous iteration.

Again g is a function completely independent from the objective function f, and can be, as before, the distance between the given solution and the best solution found so far. Tabu restrictions are sometimes used to prevent repetitions rather than reversals, as illustrated by stipulating the second condition that x_i previously changed from 1 to 0, rather than from 0 to 1. These have a rôle of preventing the repetition of a search path that leads away from a given solution. By contrast, restrictions that prevent reversals help to prevent a return to a previous solution. Hence tabu restrictions vary according to whether they are defined in terms of reversals or duplications of their associated attributes.

- <u>Tabu list size</u>

The choice of tabu list size is crucial; if its value is too small, cycling may occur in the search process, while if its value is too large, appealing moves may be forbidden, leading to the exploration of lower quality solutions and producing a larger number of iterations to find the solution desired. Empirically, tabu list sizes that provide

Table 3.2: *Illustrative rules for tabu list size.*

Static Rules

- Choose s to be a constant such as $s = 7$ or $s = \sqrt{p}$, where p is the problem's dimension.

Dynamic Rules

- *Simple dynamic*: Choose s to vary randomly or by systematic pattern, between bounds s_{min} and s_{max} with $s_{min} = 5$ and $s_{max} = 11$ or $s_{min} = 0.9\sqrt{p}$ and $s_{max} = 1.1\sqrt{p}$.
- *Attribute-dependent dynamic*: Choose s as in the simple dynamic rule, but determine s_{min} and s_{max} to be larger for attributes that are more attractive (based for example on quality or influence considerations).

good results often grow with the size of the problem. However, there appears to be no single rule that gives good sizes for all classes of problems.

Rules to determine s, the tabu list size, are classified as *static* or *dynamic.* Static rules choose a value for s that remains fixed throughout the run; dynamic rules allow the value of s to vary. Table 3.2 gives examples of these rules. The values of 7 and $\sqrt{p}$ (where p is the dimension of the problem) used in this table are only suggestive, and represent parameters whose preferred values should be set by experiment for a particular class of problems. But usually researchers use values between 7 and 20, or between $0.5\sqrt{p}$ and $2\sqrt{p}$, and in fact these values appear to work well for a large variety of problems.

Practical experience indicates that dynamic rules are typically more robust than static ones (Glover et al. 1993). This is probably the reason why many researchers prefer to use non-fixed values for the tabu list size. For example (Taillard 1991), in order to solve the quadratic assignment problem, selected the size randomly from the interval $[0.9\,p, 1.1\,p]$ and kept it constant for $2.2\,p$ iterations before selecting a new size by the same process. (Laguna et al. 1991) have effectively used dynamic rules that depend on both attribute type and quality, while a class of dynamic rules based on moving gaps was also used effectively by (Chakrapani and Skorin–Kapov 1993). Finally (Laguna and Glover 1993) systematically varied the list size over three different ranges (small, medium and large), in order to solve telecommunications bandwidth packing problems.

- **Aspiration criteria**

As mentioned above, the rôle of aspiration criteria is to determine when tabu restrictions can be overridden, and make a tabu move admissible. The appropriate use of such criteria can be very important for enabling a TS method to achieve its best performance level. The simplest and most common aspiration criterion is the one that allows a tabu move to be selected if it leads to a solution better than the best found so far. This criterion remains widely used, but other aspiration criteria can also prove effective for improving the search. The literature on such criteria identifies two types: *move aspirations* and *attribute aspirations.* When a move aspiration criterion is satisfied, a move's tabu classification is revoked, while when an attribute aspiration criterion is satisfied, an attribute's tabu active status is revoked.

Suppose that a trial solution X^{trial} is generated by a tabu move. The following is a list of the most common criteria for determining the admissibility of this solution:

- *Aspiration by default*: If all available moves are classified tabu, and are not rendered admissible by some other aspiration criteria, the "least tabu" move is selected. For example you can choose the move that loses its tabu status by the least increase in the value of the objective function.

- *Aspiration by objective (global form)*: A move aspiration is satisfied, permitting X^{trial} to be a candidate for selection, if $f(X^{trial}) > c_{best}$, where f is the objective function and c_{best} is the best solution found so far.

- *Aspiration by objective (regional form)*: Subdivide the search space into regions R. Let $c_{best}(R)$ denote the minimum $f(X)$, with $X \in R$. Then for $X^{trial} \in R$, a move aspiration is satisfied if $f(X^{trial}) > c_{best}(R)$.

- *Aspiration by search direction*: Let $direction(e) = improving$ if the most recent move containing the reverse attribute of e, $\bar{e}$, was an improving move, $direction(e) = nonimproving$ otherwise. An attribute aspiration for e is satisfied, making e admissible, if $direction(e) = improving$ and the current trial move is an improving move, i.e., $f(X^{trial}) > f(X^{now})$.

- *Aspiration by influence*: Let $influence(e) = 0$ or 1 according to whether the move that establishes the value of $tabustart(e)$ is a low-influence or high-influence move, where $tabustart(e)$ is the starting iteration of the tabu duration for attribute e. Also let $latest(L)$, for $L = 0$ or 1, equal the most recent iteration that a move of influence level L was made. Then an attribute move for e is satisfied, if $influence(e) = 0$ and $tabustart(e) < latest(1)$. We can also have multiple $influence$ levels, $L = 0, 1, \ldots$, and then the aspiration for e is satisfied, if there is an $L > influence(e)$ such that $tabustart(e) < latest(L)$.

The first of the above criteria is rarely applicable, but is understood automatically to be part of any tabu search procedure. Also the last two criteria are attribute aspirations rather than move aspirations.

The list above gives the simplest and most well-known aspiration criteria. For applications that are more complex we might need more complicated criteria, one of which is *aspiration by strong admissibility.*

Definition 3.1. A move is *strongly admissible* if it is admissible for selection and does not rely on aspiration criteria to qualify for admissibility; or if it qualifies for admissibility based on global aspiration by objective, by satisfying $f(X^{trial}) > c_{best}$.

- *Aspiration by strong admissibility*: Let $last_{nonimprovement}$ equal the most recent iteration that a nonimproving move was made, and let $last_{stronglyadmissible}$ equal the most recent iteration that a strongly admissible move was made. Then if $last_{nonimprovement} < last_{stronglyadmissible}$, we can reclassify every improving tabu move to be a candidate for selection.

With the inequality $last_{nonimprovement} < last_{stronglyadmissible}$ two facts are clear: that a strongly admissible improving move has been made since the last nonimproving move, and that the search is currently generating an improving sequence. Thus this type of aspiration ensures that the method will always proceed to a local optimum whenever an improving sequence is created that contains at least one strongly admissible move.

- <u>**Parallel processing**</u>

To speed up TS, again consider the case of parallel runs. According to (Glover et al. 1993), concurrent examination of different moves from a neighbourhood often makes it possible to reach a speed-up factor close to

$$\text{ideal speed-up} = \frac{T_n + T_u}{\frac{T_n}{P} + T_u}, \tag{3.48}$$

where P is the number of processors, T_n is the sequential time to perform the computation to be parallelised and T_u is the time associated with the non-parallelisable part of the algorithm. For example T_n may correspond to the time spent in evaluating the neighbourhood and T_u may correspond to the update of the information structure when a move is performed. Using such a technique, significant speed-up can be achieved for many problems.

In more detail, the above parallelisation idea is to divide the neighbourhood into P parts each about the same size, and evaluate each of these parts on a different processor. Having done that, every processor computes the values of the moves that are attributed to it. Then it communicates to the others its proposed move (the best move it found) and receives $(P-1)$ moves proposed from the other processors. Every processor chooses and performs the best move proposed to (or by) it. If there are many proposed moves that are of the same quality, the processors will retain, for example, only the move that was proposed by the processor that has not had an accepted proposition for the greatest number of iterations.

Another natural and simpler parallelisation process is to perform many independent searches at a time, each starting with a different initial solution and/or using a different set of parameters. Because of the fact that we have no restrictions on this process, we can apply it for every problem, and surprisingly this straightforward type of parallelisation has proven efficient for a number of processors not exceeding a few dozen. (Taillard 1989; Taillard 1990; Taillard 1991) applied this method in three different types of problems very successfully.

3.5.3 TS summary

Experiments have shown that TS is able to obtain results that match or surpass the best known outcomes using other methods in a variety of optimisation problems. At the same time, it is apparent that studies to date have only taken the first steps in exploring this potential. Many more applications remain to be undertaken, and many new possibilities for refining the basic processes of the method remain to be tested. The most interesting thing about TS to me is that it appears to be almost unknown in the statistics community: none of the 20 papers and book chapters I was able to find on TS through an extensive electronic search are in the statistics literature.

3.6 Summary of all optimisation methods studied

I conclude this chapter with a list of all optimisation methods studied in this dissertation, together with a brief description of each and an indication of how they are related to each other.

- *Local search* (greedy algorithm that only accepts uphill moves);
- *Simulated annealing* (SA; algorithm with neighbourhood structure and flexible rule based on temperature for sometimes accepting downhill moves);
- *Messy SA* (hybrid between SA and GA: first use neighbourhood structure motivated by GA ideas, then acceptance rule same as SA);
- *Threshold acceptance* (TA; a simplified version of SA, using a threshold value instead of temperature);
- *Genetic algorithm* (GA; method with selection mechanisms (crossover, mutation) instead of neighbourhood structure);
- *Genetic local search* (hybrid between local search and GA, using local search to create the initial population);
- *Genetic simulated annealing* (hybrid of SA and GA: first SA to generate population, then GA from there), messy SA and genetic SA are like two sides of the same coin;
- *Genitor algorithm* (modification of GA in which the fittest parent is included in the next generation along with a single child);
- *CHC adaptive search* (modification of GA in which (1) a version of uniform crossover is used instead of simple crossover and (2) the two best are chosen from both parents, both children and (3) the algorithm restarts itself when it gets stuck); and
- *Tabu search* (TS; algorithm with neighbourhood structure, three phases (preliminary search, intensification, diversification), and list of forbidden moves to prevent circularity).

See Table 5.3 for an example of the inputs required by the three main methods I will examine in the remainder of the dissertation: SA, GA, and TS.

Chapter 4

Results in the case $p = 14$

Everything should be made as simple as possible, but not simpler.

— Albert Einstein

4.1 Introduction

The most straightforward way to compare optimisation methods is to create a test-case in which the truth about all 2^p models is known (up to small Monte Carlo uncertainty), so that the actual quality of subsets discovered by any given optimisation method may be ascertained. As noted in Chapter 1, I chose to do this by performing three full enumerations[1] of the estimated expected utility of all $2^p = 16,384$ possible subsets of the $p = 14$ variables chosen in the Rand sickness scale for pneumonia (Table 1.2), in which each estimate of the expected utility—equation (2.5)—was based on $N = 500$ random splits (this choice of N was sufficient to yield a Monte Carlo standard error for each expected utility estimate of only about US\$0.05, which is small enough to reliably identify the good models).

Section 4.2 presents full-enumeration results for the first of the three repetitions of the process. In Section 4.3 I explore the geometry of the solution space, and Section 4.4 discusses the optimal choice of N during the stochastic optimisation runs. In Section 4.5 I present some preliminary results comparing five of the most promising optimisation methods described in Chapter 3: a genetic algorithm (GA), messy simulated annealing (MSA), simulated annealing (SA), tabu search (TS), and threshold acceptance (TA). Sections 4.6 and 4.7 describe the design and analysis of a large simulation experiment on the three leading contenders to emerge from the preliminary work: GA, SA, and TS. In Section 4.8 I conclude the chapter with the

[1] Each of these runs took 38 days of CPU time at 400 Unix MHz to complete.

results of four sensitivity analyses: (1–2) the two additional full-enumeration runs, one with a different random number seed to see how stable the findings were and one with a different choice of the proportion of data points used in the modelling and validation samples; (3) an analysis varying the penalties and rewards for prediction accuracy and marginal costs per variable, and (4) a study of the effects of including interaction terms in addition to main effects in the logistic regression modelling.

4.2 Full enumeration results

In the first full-enumeration run I used a modelling sample of $\frac{n_M}{n} = \frac{2}{3}$ of the data (1665 patients), and the remaining $\frac{n_V}{n} = \frac{1}{3}$ (867 patients) for the validation sample, and I performed $N = 500$ random splits into the modelling and validation part in each model. (All of the programming in this dissertation was in C (Kelly and Pohl 1995); the analysis of the results was conducted using S+ (Becker et al. 1993) and Stata (Stata 1997).) Clinical experts in the US and UK were asked for reasonable values of the data collection marginal costs (c_j), as well as of the penalty and reward factors in the predictive utility (the constants $C_{11}, C_{12}, C_{21}, C_{22}$ in Table 2.1), as described in Chapter 2. Data collection marginal costs were estimated as numbers of minutes of abstraction time, at roughly \$20/hour. Table 4.1 shows the costs of all 14 variables together with their correlation with 30–day death (a measure of how well they predict death within 30 days of admission).

Figure 4–1 presents parallel boxplots (Tukey 1977) of the estimated expected utilities of the 16,384 models in the $p = 14$ case as a function of k, the number of predictors in each model. The globally optimal model has four of the original 14 Rand variables—systolic blood pressure score, blood urea nitrogen (BUN), APACHE II coma score, and shortness of breath day 1, which are marked with two asterisks in Table 4.1—and achieves an estimated expected utility of -7.89 ± 0.05. Several conclusions are evident from a detailed examination of this figure and the data on which it is based, as follows.

- The trace of the median expected utilities (the white lines in the middle of the boxes) as a function of k clearly shows the tradeoff between data collection cost and predictive accuracy: for small k the models do not cost very much but predict poorly, and for large k the predictions are excellent but the cost is too high, so that the best models are in the middle. In particular the full 14–variable Rand scale is highly inefficient (and slightly worse in monetary terms than using no sickness indicators at all, i.e., predicting death at random

Table 4.1: *The 14 variables in the Rand pneumonia admission sickness scale, together with their approximate data collection costs per patient and correlation with 30–day death (CHF = congestive heart failure).*

Variable	Cost c_j (US$)	Correlation	Good?
Total APACHE II score (36-point scale)	3.33	0.39	
Age of patient	0.17	0.17	*
Systolic blood pressure score (2-point scale)	0.17	0.29	**
CHF chest X-ray score (3-point scale)	0.83	0.10	
Blood urea nitrogen (BUN)	0.50	0.32	**
APACHE II coma score (3-point scale)	0.83	0.35	**
Serum albumin score (3-point scale)	0.50	0.20	*
Shortness of breath day 1 (yes, no)	0.33	0.13	**
Respiratory distress (yes, no)	0.33	0.18	*
Septic complications (yes, no)	1.00	0.06	
Prior respiratory failure (yes, no)	0.67	0.08	
Recently hospitalised (yes, no)	0.67	0.14	
Ambulatory score (3-point scale)	0.83	0.22	
Initial temperature	0.17	−0.06	*

Note: The final column of the table is explained in the text below.

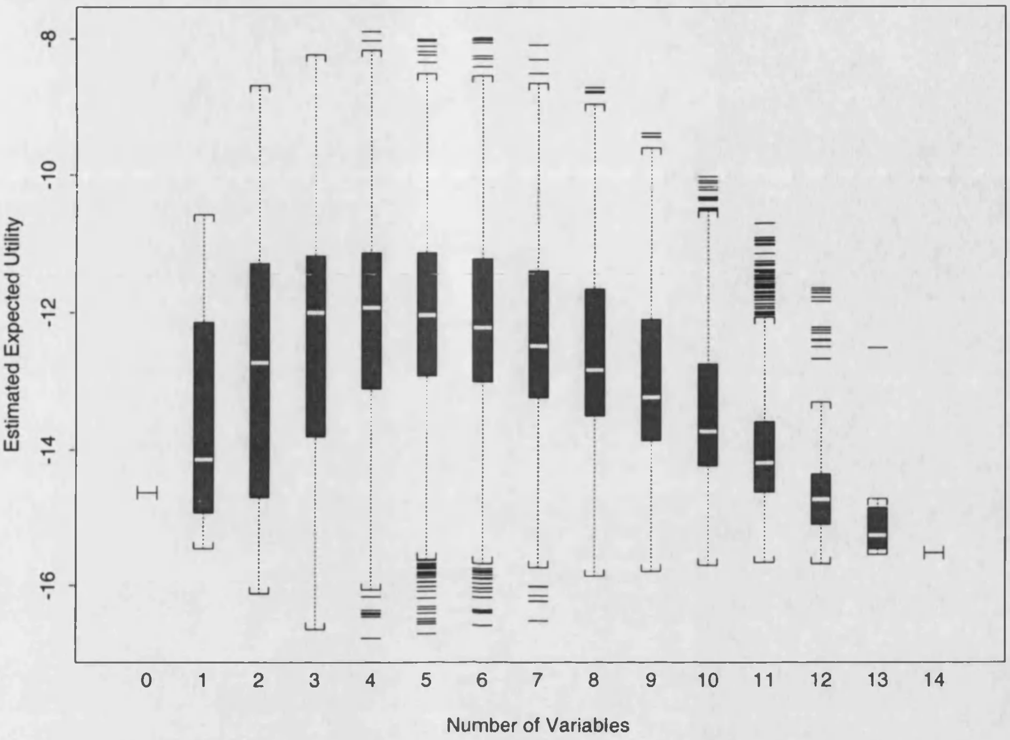
with probability 0.16).

- The 20 best models include the same 3 variables 19 or more times out of 20, and never include 5 of the other variables; the five best models are minor variations on each other, and include 4–6 variables (thus no overwhelming significance should be attached to the precise model identified by the double asterisks in Table 4.1). The eight variables which occur frequently in the 20 best models are identified with asterisks in the Table. The single best univariate predictor, the total APACHE II score, does not appear in any of the good models because it is so costly to collect—BUN and the APACHE II coma score predict death (univariately) almost as well and are much cheaper to obtain.

- The best models cost almost US$8 per patient less than the full 14–variable model, which would yield significant savings annually if the input-output approach were to be implemented on a widespread basis.

- It will be seen in Section 4.8 that these results are stable across random repetition, varying the proportions $\left(\frac{n_M}{n}, \frac{n_V}{n}\right)$, changing the utilities and data collection costs, and including or not including interaction terms in the logistic

Figure 4-1: *Estimated expected utility as a function of number of predictors retained, from the first full-enumeration run with* $p = 14$.



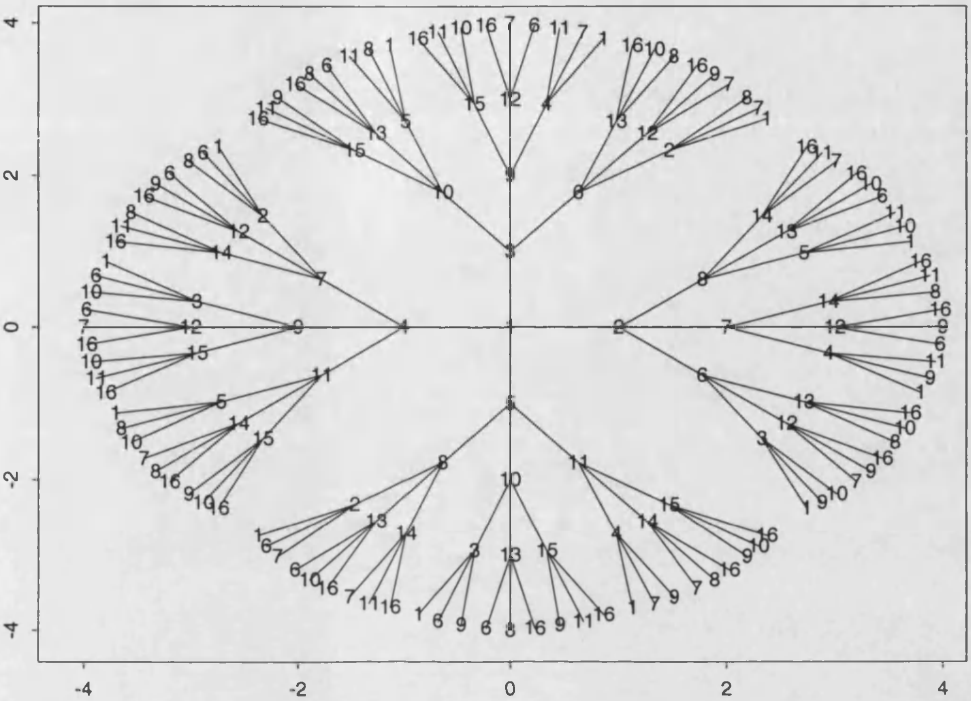
modelling. I conclude that "truth" in the 14–variable case has been identified with sufficient accuracy to serve as a basis for comparing various stochastic optimisation methods in their ability to recover that "truth" when given a limited amount of CPU time.

4.3 Geometry of the solution space

Before experimenting with the optimisation algorithms I performed a small study aiming to visualise the solution space. I wanted to have a rough idea of how smooth this space is, how many local optima it has, and how far the global optimum is from the other solutions. It is intuitively clear that it is far easier to optimise a convex function than one without this property. Evidently convexity does not hold in this problem, but I was curious to see in some sense "how far from convexity" the objective function is in this case.

Optimisation methods such as SA and TS require the specification of a neighbourhood structure across models, so that—having evaluated the quality of a given model—one can judge where best to move next in the search for the global

Figure 4-2: *Tree of adjacent models (k = 4) expanded out to four levels, with the neighbourhood structure induced by moves based on one-bit flips. The horizontal and vertical scales are arbitrary.*



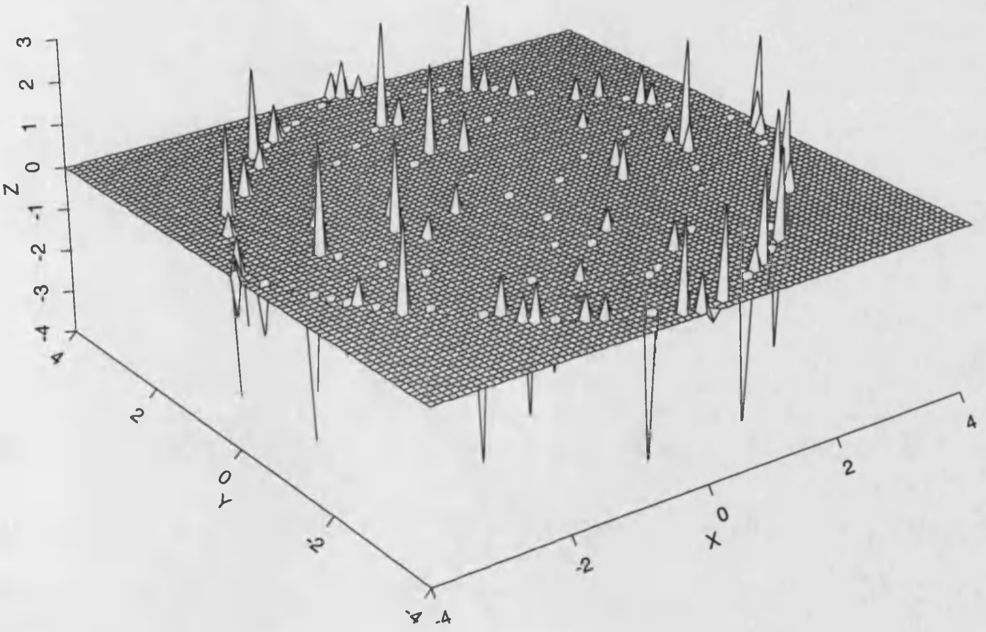
optimum. In the problem addressed here a model is a vector of p 1s and 0s specifying the presence or absence of each predictor in the subset of available variables, and a natural first choice for neighbourhood structure is based on moves which select a single bit in the binary string and flip it from 0 to 1 or vice versa (*1–bit flips*).

Whatever the neighbourhood structure, the space of all possible models can be visualised as a *tree* (Knuth 1968). Figure 4–2 shows the $2^4 = 16$ models for one particular choice of $k = 4$ variables chosen from among the 14 predictors in the Rand scale, with the tree expanded out to four levels; Table 4.2 gives the correspondence between the binary and decimal representations of the 16 models. The neighbourhood structure is evident—for example model $2/(0,0,0,1)$ is a neighbour of models $1/(0,0,0,0), 8/(1,0,0,1), 7/(0,1,0,1)$, and $6/(0,0,1,1)$. Figure 4–3 is a perspective plot of the expected utility "surface" corresponding to the tree in the previous figure. The X and Y axes match the horizontal and vertical axes in Figure 4–2; the Z axis plots the estimated expected utilities of the 16 models (from a full-enumeration exercise like that described earlier on), shifted so that the median utility is zero.

Table 4.2: *Renaming the 16 models with $p = 4$.*

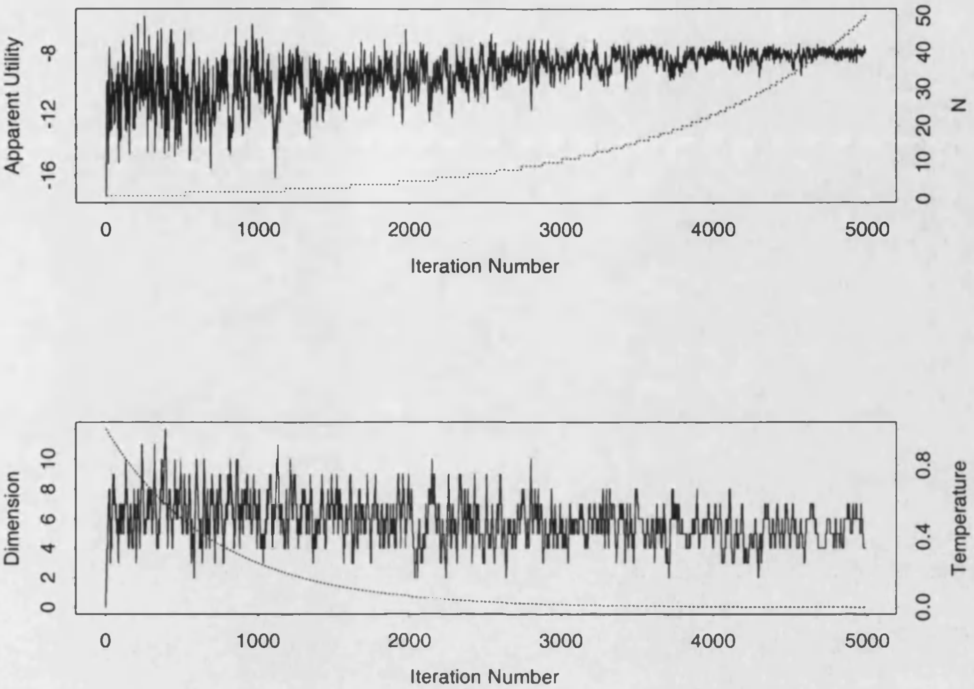
Model	0 0 0 0	0 0 0 1	0 0 1 0	0 1 0 0	1 0 0 0	0 0 1 1	0 1 0 1	1 0 0 1
Index	1	2	3	4	5	6	7	8
Model	0 1 1 0	1 0 1 0	1 1 0 0	0 1 1 1	1 0 1 1	1 1 0 1	1 1 1 0	1 1 1 1
Index	9	10	11	12	13	14	15	16

Figure 4-3: *Perspective plot of the expected utility "surface" for the 4-variable tree expanded out to four levels.*



While it is true that the quality of a given model's neighbours is sometimes similar to that of the model itself, it is also evident that adjacent models can have sharply different expected utilities, demonstrating the discontinuity of the solution space in this problem: good models do not necessarily have good models as neighbours. This has implications for the optimal search strategy—methods that spend considerable time exploring local alternatives to good models may not perform as well as methods that frequently make large jumps around the model space, but too much jumping around in an unguided way will yield poor performance as well. This is going to make the work of our optimisation algorithms difficult, since they need to be "clever" enough to manage to escape the local optima and still find the global maximum in a reasonable amount of CPU time.

Figure 4-4: *Performance of SA on a run that found the global optimum in the* $p = 14$ *case, allowing the method 24 hours of CPU time at 400 MHz.*

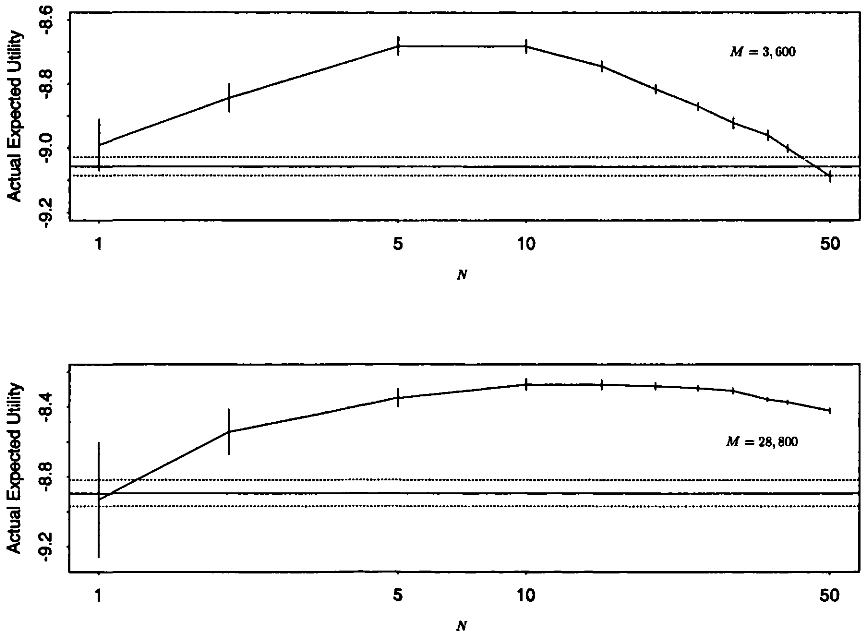


4.4 Optimal choice of N

There is a final problem to address before examining the performance of the optimisation algorithms: what is the optimal number N of utility evaluations across modelling/validation splits? As we have seen from the full enumeration runs a value like 500 gives us accurate results, but with a fixed budget of CPU time this may be a luxury that cannot be afforded. With such a budget, if N is small you can visit a lot of models, but you get a noisy estimate of how good each model is; if N is big you know how good the models really are, but you can visit less models. By using a value like 500, we cause the algorithm to visit a tiny number of models, which may well not give it enough time to find the correct maximum. On the other hand a very small value of N will give plenty of time for the algorithm to explore the solution space, but the model the algorithm will report as the best may not be the best at all. So there is another optimisation problem here, and it is intuitively clear that a value of N somewhere in the middle has to be the optimal solution.

As an instance of this phenomenon, Figure 4–4 presents an example of the performance of SA, in a run with $p = 14$ in which SA found the global optimum

Figure 4-5: *Actual expected utility as a function of N for a random-walk search strategy (the horizontal scale is logarithmic).*



solution described in Section 4.2. I used a geometric cooling schedule from a starting temperature of 1 to a final temperature of 0.001, and moves from one model to another were based on 1–bit flips. The run consisted of 4,989 iterations, with N beginning at 1 and increasing geometrically to 50, and the null model (with no predictors) was used as the starting value. Four aspects of the run are plotted: (apparent) estimated expected utility and N (the left- and right-hand vertical scales in the upper panel), and dimension k of the current model and temperature (the left- and right-hand scales in the lower panel). It is evident that from about iteration 3,000 to the end SA primarily visited good models with 3–7 predictors (the optimal range in Figure 4–1), but the method spent much of its time before that point looking at models known from the results in Section 4.2 to be inferior. This may well be (in part) because values of N that were too small were used early in the run: note, for example, that in the first 1,000 iterations (when N was at most 2) SA found several models with apparent estimated expected utility of about -6, which is much larger than the actual utility of the best models.

To explore the optimal choice of N in a simple setting, as a way of informing its choice in the main experiments below, I compared two search strategies: random-walk in model space (a) with $N = 1$, and (b) with $N > 1$. Each strategy was given a budget of M utility evaluations (which is equivalent to a CPU constraint); strategy

(a) visited M models chosen at random from the 16,384 possible models in the $p = 14$ case, with each model having its expected utility evaluated only once, whereas strategy (b) visited $\frac{M}{N}$ models at random and estimated the expected utility as an average of N evaluations across random modelling/validation splits. I made a total of 11 separate runs at each value of M, using $N = 1, 2, 5, 10, 15, 20, 25, 30, 36, 40,$ and 50. In each simulation replication the *actual* expected utility (from the full-enumeration results of Section 4.2) of the model with the maximum *apparent* expected utility across all models visited was recorded, and I repeated this exercise $m = 3,000$ times for each value of N and for M varying from 3,600 to 28,800. Monte Carlo error was estimated in the usual way for the mean of IID draws from a population, namely $\widehat{SE}(\bar{E}) = \frac{s_E}{\sqrt{m}}$, where $\bar{E}$ and s_E are the sample mean and standard deviation, respectively, of the m replications at each setting of N and M.

Figure 4–5 summarises the results for the extremes of M I examined. The roughly quadratic curves (with 95% uncertainty bands plotted as solid vertical lines) trace out the mean *actual* expected utility as a function of N across the simulation replications; the horizontal line in each plot (with 95% uncertainty bands as dotted lines) gives the results from a separate set of runs with $N = 1$ for comparison. For $M = 3,600$ the optimum N is attained between 5 and 10, with the results for $N = 50$ about as bad as those with $N = 1$. When the number of utility evaluations is increased by a factor of 8, both strategies naturally find better models and the optimal N increases to about 10 (although values of N between 10 and 30 do almost as well as $N = 10$). Even though strategies (a) and (b) are much less sophisticated than those examined in Section 4.6, I found that the results described here are a good guide to the sensible choice of N when more intelligent global optimisation methods are used instead.

4.5 Comparison of optimisation methods: preliminary results

I first completed a preliminary comparison, on the $p = 14$ case, of five stochastic optimisation methods: a genetic algorithm (GA), messy simulated annealing (MSA), simulated annealing (SA), tabu search (TS), and threshold acceptance (TA). To give all of the methods a realistically small amount of CPU time with $p = 14$ (to simulate the situation with larger p), I made a number of runs forcing each method to only use 10 or 20 minutes of CPU time at 400 Unix MHz. The neighbourhood structure for TS was based on 1–bit flips; with TA and SA I alternated 1–bit flips with a second move type: 2–bit swaps, in which a random subset of two variables is chosen and their inclusion indicators, if different, are interchanged. Results in Table 4.3 were averaged over 100 Monte Carlo repetitions and rounded to the nearest integer.

In these runs I also implemented an improvement involving adaptive choice of N: previously in a fixed–N run with (say) $N = 10$ all models were evaluated with $N = 10$. In the adaptive method (i) 20 models are chosen at random to initialise the search and evaluated with $N^* = 10$, creating a *league table* of the current 20 best models; and (ii) a new model is chosen and evaluated once. If its apparent utility would seem to place it somewhere in the current league table, the utility is evaluated for $(N^* - 1) = 9$ more random splits and the average over the N^* values is computed—if it still belongs in the league table it is added at the appropriate place; if not it is discarded. I found that this *adaptive–N^** approach was significantly better than the fixed–N approach for all optimisation methods I examined.

There is some evidence (Aarts and Korst 1989) that SA can perform particularly badly when the surface to be optimised is very rough, and this point by itself would suggest focussing on runs with large values of N for SA. It is certainly true that running SA with a constant value of $N = 5$ would produce a clearer picture of the quality of the models it manages to visit, but with a fixed budget of CPU time it would be able to visit 5 times fewer models with this approach. The adaptive–N^* method offers the benefit of not spending too much time on apparently bad models without running too large a risk of overlooking models that are actually good.

Table 4.3 presents the results of the preliminary comparison with 10 and 20 minutes of CPU time. The adaptive–N^* method was used throughout. In the version of TS studied here I chose the following user-defined settings: the simplest aspiration criterion described in Section 3.5.2, a tabu list size of 7 (I held these first two choices constant throughout the entire dissertation), r repetitions of the whole search process (where r varied from 1 to 11 as a function of N and the amount of CPU time allowed), 6 preliminary searches, 9 intensification searches, a maximum of 4 random restarts within each intensification search (a restart occurred whenever the globally best solution found so far was located), and 2 diversification searches. The version of SA in this preliminary comparison employed a geometric cooling schedule from a maximum temperature of 1.0 to a minimum of 0.1. The version of MSA used in Table 4.3 was identical to SA except with gene and allele mutation probabilities both set to 0.5. In this version of TA I varied the threshold, on the utility scale, geometrically throughout the run from an initial value of 1.0 to a minimum of 0.1. Finally, in the version of GA whose results are given in Table 4.3 the population size was 40, I used one-bit crossover with probability 0.7, the mutation probability

Table 4.3: *Preliminary comparison of GA, MSA, SA, TS, and TA. The adaptive-N^* method was used in all cases. Boldface indicates the best result in each column for each CPU time constraint.*

10 minutes CPU time

		20 Best Models Found			
		Number of	Mean (SD)	Actual Rank of	
Method	N^*	20 Actual Best	Actual Utility	Best	Worst
TS	5	**11**	**−8.36** (0.29)	4	**103**
TA	3	10	−8.43 (0.37)	2	119
SA	1	9	−8.65 (0.64)	2	1,321
MSA	3	4	−8.84 (0.51)	**1**	499
GA	2	1	−9.58 (0.64)	9	2,379

20 minutes CPU time

		20 Best Models Found			
		Number of	Mean (SD)	Actual Rank of	
Method	N^*	20 Actual Best	Actual Utility	Best	Worst
TS	15	**15**	**−8.26** (0.32)	**1**	153
TA	10	12	−8.30 (0.29)	**1**	**67**
SA	4	13	−8.32 (0.30)	**1**	102
MSA	3	8	−8.57 (0.43)	**1**	190
GA	1	3	−9.22 (0.77)	6	5,562

was 0.001, and the fitness and objective functions g and f were taken to be equal (this also remains true throughout the whole dissertation).

Each row in the Table 4.3 represents the best of eight runs, corresponding to $N^* = \{1, 2, 3, 4, 5, 10, 15, 20\}$. In keeping with the likely use of this method in health policy, in which a list of the b best models would be presented to decision-makers for a check on clinical face-validity, I examined three summaries of how well each method recovered the $b = 20$ best models from the full-enumeration exercise: (1) how many of the actual 20 best models were in each method's announced list of 20 best, and the actual ranks of the (2) best and (3) worst models in the apparent 20 best. (In column 4 of Table 4.3 I also report the mean and standard deviation of the actual utility of the 20 best models found by each method.) In these preliminary comparisons (and others not shown here for reasons of space) I found that TS was the overall best method in this problem (routinely able to find about 75% of the 20 best models in only 20 minutes of CPU time), with TA and SA not far behind; MSA came in an unimpressive fourth, and GA decisively brought up the rear. Given that differences of 0.20 or more on the utility scale are large in practical terms in this problem (because of the financial implications of such differences), the utility results in Table 4.3 convey a similar message.

By looking at the geometry of the solution space it was clear in Section 4.3 that good methods in this problem need to strike a compromise between respecting the local neighbourhood structure and making bold jumps around the model space. Careful examination of GA and TS results indicates that the cross-over operation inherent in the version of GA used here makes insufficient use of the modest amount of continuity present in this problem, while TS appears to achieve a happy balance between local exploration of good models and occasional leaps into fruitful new territory. It is the diversification stage of TS that appears to give it the edge over GA and SA in this problem.

4.6 A simulation experiment with $p = 14$

On the basis of the preliminary results I chose (a) to explore a wider variety of implementations of GA, in an attempt to overcome its poor performance, and (b) to focus only on GA, SA, and TS, because the other two methods either did not perform well or are minor variations on the three main approaches. There is a surprising lack in the literature of advice on the best input settings to choose for GA, SA, and TS, so I decided to conduct a large simulation experiment in the 14–variable case to explore the effects of the input settings. As with the preliminary runs in the previous section, I wanted to restrict each method to a realistically small amount of CPU time; for the main experiment I chose 20 minutes at 400 Unix MHz.

It was also clear from the preliminary results that a single run of any given method with any particular set of inputs gave a quite noisy estimate of the performance, so I made 30 runs, with different starting random seeds, for each input setting, and averaged the results. I used the adaptive–N^* method, which has already been shown to clearly dominate the fixed–N approach, and I varied N^* as another parameter in the simulation design.

The main outcome variable I examined was the percentage p_{20} of the actual 20 best models found with each set of inputs (other obvious outcomes such as the actual utility of the best models found correlated strongly with p_{20}, as is clear from Table 4.3). Since Figure 4–1 shows that there are a number of models whose performance is close to optimal, I chose p_{20} to mimic the decision-making reality that in practice, instead of giving the health policy colleagues with whom I would be working a single best model, it would be better to give them the q best models (for a value of q, with $p = 14$, like 20) and let them choose on grounds that were partly statistical and partly clinical. The number of repetitions of the experiment at each input setting, 30, was chosen to make the Monte Carlo standard error small enough so that differences of 0.05 in p_{20} between two methods or input settings would be detectable.

4.6.1 Tabu search

With TS in this problem there are six user inputs to vary:

- r, the total number of repetitions of the algorithm (this varied from 1 to 6 in the experiment);
- N^*, the maximum number of random modelling/validation splits on which the estimated expected utility is based (the actual number was either 1 or N^* in accordance with the adaptive method). This varied from 2 to 20;
- l, the number of preliminary searches per repetition (this varied from 2 to 21);
- i, the number of intensification searches per repetition (this varied from 2 to 40);
- t, the maximum number of random restarts in each intensification search (this varied from 0 to 8); and
- d, the number of diversification searches per repetition (this varied from 1 to 15).

In all parts of the experiment, for all three optimisation methods, the final ranges of the inputs used in the simulation study were chosen from preliminary runs to span a range of performance from bad to good to bad again (as far as that input was concerned).

A full factorial across all six of these inputs is not possible because many of them lead to CPU times much greater than the target of 1,200 seconds. By trial and error I was able to find 49 combinations of input settings, each of which took approximately 20 minutes of CPU time. The actual CPU time varied by input settings from a mean across the 30 runs of 1075 sec to 1575 seconds, so I have calculated both raw summaries and results adjusted (via regression) for differences in CPU time.

Tables 4.4–5 summarise the results of the simulation study for TS. Its performance varied noticeably according to input settings, from an adjusted mean for p_{20} of 39% to 65%. I tried fitting linear models to the data in these tables to evaluate the effects of each of the TS inputs on performance, but these effects proved complicated to quantify because of strong and high-order interactions among the inputs. Only one clear pattern emerges: good runs tend to have input values in the middle, and bad runs tend to have values at the extremes, of the ranges used. It is fairly impressive that the best input settings of TS can find about 2/3 of the best models among a collection of 16,384 with only 20 minutes of CPU time.

4.6.2 Simulated annealing

With simulated annealing in this problem there are five user inputs to vary:

- r, the total number of iterations (this varied in the experiment from 130 to 2500);

- N^*, as in TS (this again varied from 2 to 20);

- T_0 and T_f, the initial and final values of the temperature (these varied across the five settings $(T_f, T_0) = (10.0, 1.0)$, $(10.0, 0.1)$, $(2.5, 0.1)$, $(1.0, 0.1)$, $(0.5, 0.05)$); and

- *sc*, the schedule used to decrease the temperature (1 = straight, 2 = geometric, 3 = reciprocal, 4 = logarithmic).

By trial and error I was able to find 108 combinations of input settings (almost a full factorial), each of which took approximately 1,200 seconds. The actual CPU time again varied by input settings, this time from a mean (across the 30 runs) of 986 to 1471 seconds, so as before I calculated both raw summaries and results adjusted (via regression) for differences in CPU time.

The performance of SA in this problem, summarised in Tables 4.6–8, varied dramatically according to input settings (substantially more than with TS), from an adjusted mean for p_{20} of 0% to 56%. I again tried fitting linear models, but the effects again proved complicated to quantify because of strong and high-order interactions among the inputs. Some conclusions emerging from the tables are as follows.

- SA's performance was disappointing when compared with that of TS: the best SA runs were below the median TS results. (However, in Chapter 6 I report results of an improved SA method that are much better.)

- Small values of N (up to 10) appear best.

Table 4.4: *Results of the simulation study for TS with $p = 14$ (part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of p_{20}.*

							p_{20} (% of 20 Actual Best Models Found)		
						Mean CPU	Raw	Raw	Adjusted
r	N^*	l	i	t	d	Time (sec)	Mean	SD	Mean
1	10	10	6	3	3	1158	0.631 (0.020)	0.109	0.649
5	6	2	7	2	2	1362	0.641 (0.016)	0.089	0.635
2	10	2	4	6	4	1313	0.635 (0.016)	0.090	0.634
4	9	2	4	1	2	1386	0.636 (0.017)	0.091	0.627
2	5	14	15	5	1	1243	0.611 (0.016)	0.089	0.619
1	10	9	12	3	3	1273	0.615 (0.015)	0.084	0.619
1	14	11	2	0	1	1253	0.611 (0.019)	0.106	0.618
6	7	2	4	2	2	1574	0.650 (0.017)	0.095	0.618
5	5	9	3	3	2	1300	0.615 (0.018)	0.099	0.616
5	8	2	4	1	3	1500	0.631 (0.013)	0.070	0.608
3	10	2	4	2	2	1321	0.605 (0.013)	0.069	0.603
2	5	9	10	3	14	1200	0.590 (0.019)	0.106	0.603
1	14	2	11	0	1	1247	0.595 (0.019)	0.106	0.602
1	10	12	6	1	2	1230	0.590 (0.024)	0.134	0.599
3	5	12	9	4	3	1318	0.600 (0.018)	0.096	0.599
5	5	4	7	3	2	1326	0.600 (0.018)	0.098	0.598
1	20	4	6	4	2	1258	0.591 (0.032)	0.174	0.597
1	14	3	5	6	2	1291	0.593 (0.021)	0.113	0.595
5	5	3	9	3	2	1377	0.601 (0.015)	0.083	0.593
6	6	2	4	2	2	1256	0.586 (0.018)	0.100	0.593
2	10	2	5	4	5	1332	0.595 (0.021)	0.116	0.592
1	7	11	18	5	10	1223	0.581 (0.015)	0.080	0.592
3	5	7	9	3	7	1239	0.583 (0.017)	0.091	0.591
4	10	10	10	0	1	1350	0.595 (0.019)	0.105	0.590
1	6	11	30	5	2	1228	0.578 (0.020)	0.112	0.588
4	5	6	9	4	3	1345	0.591 (0.013)	0.070	0.587
1	10	21	7	1	1	1200	0.571 (0.017)	0.093	0.584
1	6	20	14	6	2	1074	0.555 (0.021)	0.114	0.584
6	5	3	6	2	3	1325	0.583 (0.015)	0.081	0.581
2	9	7	2	2	4	1217	0.568 (0.019)	0.103	0.579
2	12	3	7	2	3	1521	0.601 (0.019)	0.103	0.576
5	4	6	7	2	4	1278	0.560 (0.019)	0.102	0.563
6	4	4	7	2	3	1253	0.548 (0.022)	0.119	0.555
4	5	6	9	0	4	1328	0.556 (0.017)	0.095	0.554
1	18	2	3	6	1	1253	0.546 (0.034)	0.185	0.553
1	7	12	28	8	5	1411	0.561 (0.015)	0.084	0.549
1	5	17	38	0	15	1248	0.541 (0.018)	0.096	0.549

Table 4.5: *Results of the simulation study for TS with $p = 14$ (part 2).*

							p_{20} (% of 20 Actual Best Models Found)		
						Mean CPU	Raw	Raw	Adjusted
r	N^*	l	i	t	d	Time (sec)	Mean	SD	Mean
6	3	6	9	3	4	1343	0.536 (0.016)	0.089	0.532
1	5	19	35	5	7	1298	0.530 (0.016)	0.087	0.531
6	4	5	3	2	8	1306	0.526 (0.022)	0.122	0.527
1	5	20	40	8	10	1319	0.526 (0.016)	0.089	0.525
6	6	4	2	2	3	1319	0.523 (0.022)	0.122	0.522
1	20	3	5	0	2	1240	0.475 (0.027)	0.147	0.483
2	10	4	2	1	6	1300	0.441 (0.027)	0.148	0.442
2	10	4	8	1	2	1541	0.446 (0.035)	0.193	0.418
3	2	15	35	8	2	1448	0.430 (0.020)	0.110	0.413
1	15	5	3	1	6	1191	0.398 (0.047)	0.257	0.412
1	15	6	12	1	5	1523	0.431 (0.039)	0.214	0.406
5	8	4	1	1	3	1327	0.395 (0.028)	0.155	0.392

- $(T_f, T_0) = (1.0, 0.1)$ and $(0.5, 0.05)$, the lowest initial and final temperatures, appear best.
- The reciprocal and logarithmic schedules appear best in this problem.
- $(T_f, T_0) = (10.0, 1.0)$, large N, and the straight schedule perform badly.

4.6.3 Genetic algorithm

With GA in this problem there are six user inputs to vary:

- r, the total number of repetitions (this varied from 2 to 237);
- N^*, as in TS and SA (this varied for GA from 2 to 15);
- n, the population size (I used the three settings 30, 50, and 80);
- (c, p_c), the crossover strategy (I used $c = 1$ (simple), 2 (uniform), and 3 (highly uniform crossover). With the first strategy I used a crossover probability $p_c = 0.88$ when the population size was 30, 0.5 when the population size was 50 and 0.3 when the population size was 80.
- (e, p_m), elitist or non-elitist strategy. (In the case of the elitist strategy with highly uniform crossover there is a possibility for the new population to be

Table 4.6: *Results of the simulation study for SA with $p = 14$ (part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of p_{20}.*

						p_{20} (% of 20 Actual Best Models Found)		
					Mean CPU	Raw	Raw	Adjusted
r	N^*	T_f	T_0	sc	Time (sec)	Mean	SD	Mean
1050	4	1	0.1	3	1146	0.538 (0.021)	0.113	0.555
1000	5	0.5	0.05	1	1087	0.515 (0.016)	0.088	0.551
860	5	0.5	0.05	2	1134	0.523 (0.023)	0.125	0.544
900	5	2.5	0.1	3	1140	0.520 (0.021)	0.116	0.539
1120	3	1	0.1	3	1015	0.470 (0.023)	0.127	0.530
240	10	0.5	0.05	4	1113	0.501 (0.018)	0.097	0.530
240	10	1	0.1	4	1085	0.490 (0.027)	0.150	0.527
900	5	1	0.1	3	1138	0.505 (0.018)	0.098	0.524
1400	4	1	0.1	2	1191	0.521 (0.026)	0.145	0.524
500	10	10	0.1	3	1278	0.550 (0.021)	0.115	0.523
600	10	1	0.1	2	1279	0.550 (0.022)	0.118	0.523
450	10	1	0.1	3	1315	0.556 (0.021)	0.115	0.512
1110	5	10	0.1	3	1326	0.555 (0.025)	0.136	0.512
600	10	0.5	0.05	1	1394	0.576 (0.021)	0.116	0.511
430	10	0.5	0.05	2	1383	0.566 (0.017)	0.093	0.505
130	20	0.5	0.05	4	1216	0.506 (0.019)	0.105	0.500
150	15	0.5	0.05	4	1059	0.450 (0.023)	0.124	0.496
300	10	0.5	0.05	3	1214	0.500 (0.018)	0.099	0.494
500	5	10	0.1	4	1172	0.483 (0.018)	0.101	0.492
450	10	2.5	0.1	3	1352	0.541 (0.016)	0.089	0.490
1600	2	0.5	0.05	1	1028	0.423 (0.019)	0.103	0.479
1550	3	1	0.1	2	1146	0.461 (0.024)	0.129	0.479
240	10	2.5	0.1	4	1086	0.441 (0.026)	0.142	0.479
130	20	10	0.1	4	1161	0.461 (0.021)	0.117	0.474
630	5	1	0.1	4	1347	0.521 (0.022)	0.122	0.472
1200	5	2.5	0.1	2	1139	0.451 (0.021)	0.114	0.471
1500	2	1	0.1	3	1068	0.425 (0.026)	0.140	0.468
350	10	10	0.1	4	1467	0.556 (0.018)	0.100	0.467
520	5	0.5	0.05	4	1226	0.476 (0.019)	0.105	0.467
1500	2	2.5	0.1	3	1042	0.410 (0.024)	0.132	0.461
150	15	1	0.1	4	1089	0.425 (0.021)	0.117	0.461
290	15	0.5	0.05	1	1300	0.495 (0.029)	0.157	0.461
130	20	2.5	0.1	4	1193	0.455 (0.020)	0.112	0.456
1200	5	1	0.1	2	1158	0.443 (0.022)	0.122	0.456
1360	2	0.5	0.05	2	1053	0.408 (0.023)	0.128	0.456
1400	5	1	0.1	1	1161	0.441 (0.017)	0.092	0.454
170	15	0.5	0.05	3	1258	0.471 (0.025)	0.136	0.451

Table 4.7: *Results of the simulation study for SA with $p = 14$ (part 2).*

						p_{20} (% of 20 Actual Best Models Found)		
					Mean CPU	Raw	Raw	Adjusted
r	N^*	T_f	T_0	sc	Time (sec)	Mean	SD	Mean
150	15	2.5	0.1	4	1141	0.428 (0.024)	0.131	0.447
200	15	10	0.1	4	1440	0.525 (0.023)	0.126	0.445
1600	4	1	0.1	1	1194	0.443 (0.024)	0.134	0.444
1900	2	2.5	0.1	2	1112	0.408 (0.019)	0.105	0.436
700	5	0.5	0.05	3	1319	0.476 (0.022)	0.118	0.436
1900	2	1	0.1	2	1140	0.416 (0.022)	0.118	0.435
280	15	10	0.1	3	1374	0.490 (0.031)	0.169	0.431
1770	3	1	0.1	1	1168	0.421 (0.017)	0.095	0.431
630	5	2.5	0.1	4	1468	0.516 (0.022)	0.121	0.427
600	10	1	0.1	1	1062	0.381 (0.033)	0.183	0.426
680	4	1	0.1	4	1307	0.460 (0.025)	0.136	0.424
910	2	2.5	0.1	4	1038	0.363 (0.027)	0.149	0.416
130	20	1	0.1	4	1177	0.405 (0.024)	0.130	0.412
1400	5	10	0.1	2	1237	0.421 (0.024)	0.130	0.408
250	15	0.5	0.05	2	1393	0.471 (0.030)	0.165	0.407
200	15	1	0.1	3	1245	0.420 (0.022)	0.121	0.404
760	3	1	0.1	4	1207	0.406 (0.029)	0.160	0.403
600	10	2.5	0.1	2	1212	0.405 (0.027)	0.148	0.400
190	20	2.5	0.1	3	1346	0.448 (0.028)	0.155	0.399
630	10	10	0.1	2	1153	0.381 (0.023)	0.126	0.396
200	15	2.5	0.1	3	1174	0.386 (0.020)	0.110	0.394
145	20	0.5	0.05	3	1273	0.418 (0.028)	0.156	0.393
910	2	1	0.1	4	1046	0.338 (0.020)	0.112	0.388
1640	2	10	0.1	3	1099	0.351 (0.023)	0.128	0.384
2100	2	1	0.1	1	1163	0.366 (0.024)	0.130	0.378
170	20	0.5	0.05	2	1271	0.398 (0.024)	0.134	0.374
1160	2	0.5	0.05	3	1128	0.350 (0.028)	0.151	0.373
1000	2	0.5	0.05	4	1162	0.360 (0.025)	0.136	0.372
200	20	0.5	0.05	1	1348	0.420 (0.022)	0.122	0.370
2100	2	10	0.1	2	1182	0.363 (0.017)	0.095	0.368
350	15	1	0.1	2	1338	0.406 (0.027)	0.150	0.360
190	20	1	0.1	3	1399	0.420 (0.029)	0.161	0.353
180	20	10	0.1	3	1265	0.371 (0.027)	0.150	0.349
410	15	1	0.1	1	1209	0.350 (0.035)	0.191	0.346
1050	2	10	0.1	4	1194	0.333 (0.027)	0.147	0.334
410	15	2.5	0.1	2	1471	0.418 (0.031)	0.170	0.327
240	20	1	0.1	2	1390	0.368 (0.026)	0.140	0.304
1450	5	2.5	0.1	1	1077	0.263 (0.021)	0.117	0.303
240	20	2.5	0.1	2	1158	0.290 (0.025)	0.137	0.303

Table 4.8: *Results of the simulation study for SA with $p = 14$ (part 3).*

						p_{20} (% of 20 Actual Best Models Found)		
					Mean CPU	Raw	Raw	Adjusted
r	N^*	T_f	T_0	sc	Time (sec)	Mean	SD	Mean
360	15	10	0.1	2	1201	0.300 (0.027)	0.147	0.299
2100	2	2.5	0.1	1	1091	0.250 (0.019)	0.104	0.285
260	20	1	0.1	1	1197	0.256 (0.028)	0.152	0.257
750	10	2.5	0.1	1	1095	0.220 (0.027)	0.149	0.254
1600	5	10	1	4	1120	0.210 (0.019)	0.105	0.235
680	10	10	1	4	986	0.151 (0.021)	0.117	0.221
1800	5	10	0.1	1	1261	0.228 (0.025)	0.137	0.207
350	15	2.5	0.1	1	1060	0.156 (0.022)	0.121	0.202
2350	2	10	1	4	1193	0.191 (0.017)	0.092	0.193
230	20	10	0.1	2	1098	0.160 (0.024)	0.132	0.193
2400	2	10	1	3	1232	0.193 (0.017)	0.095	0.182
260	20	2.5	0.1	1	1075	0.108 (0.019)	0.105	0.149
2360	2	10	0.1	1	1203	0.145 (0.017)	0.095	0.143
1900	5	10	1	2	1314	0.180 (0.021)	0.115	0.141
2450	2	10	1	2	1243	0.153 (0.015)	0.082	0.138
2100	5	10	1	1	1361	0.186 (0.022)	0.121	0.132
2500	2	10	1	1	1267	0.155 (0.018)	0.101	0.132
870	10	10	1	3	1123	0.105 (0.015)	0.084	0.129
820	10	10	0.1	1	1087	0.085 (0.016)	0.086	0.121
320	20	10	1	4	1093	0.086 (0.015)	0.080	0.121
840	10	10	1	2	1097	0.086 (0.022)	0.118	0.120
470	15	10	1	3	1086	0.075 (0.012)	0.067	0.112
260	20	10	0.1	1	994	0.040 (0.009)	0.051	0.107
1800	5	10	1	3	1407	0.171 (0.020)	0.107	0.102
480	15	10	1	2	1105	0.070 (0.014)	0.077	0.100
580	15	10	0.1	1	1210	0.098 (0.024)	0.133	0.094
360	15	10	1	4	1147	0.068 (0.014)	0.074	0.085
1000	10	10	1	1	1202	0.076 (0.013)	0.073	0.075
360	20	10	1	2	1128	0.050 (0.011)	0.061	0.073
340	20	10	1	3	1133	0.036 (0.012)	0.064	0.058
360	20	10	1	1	1163	0.033 (0.009)	0.051	0.044
800	15	10	1	1	1426	0.055 (0.010)	0.057	0.000

Note: Negative adjusted mean p_{20} values have been truncated at 0 but appear in rank order corresponding to their untruncated values.

exactly the same as the previous one. In this case I decrease d, the certain Hamming distance that the two strings are away from each other; usually the starting value of d is $\frac{p}{4}$, where p is the size of the strings. If d becomes negative then I perform a kind of diversification. I replace the current population with n (the population size) copies of the best member of the previous population, and for all but one member of the new population I flip half of the bits at random. Then the algorithm is restarted.) In the elitist strategy you compare the offspring with the parents and choose the best two among the four; with the non-elitist you always choose the offspring. With the elitist strategy mutation is not performed; in the case of the non-elitist strategy mutation occurs with probability $p_m = 0.01$; and

- k: At the end of each repetition, I either clear the population and randomly generate a new one ($k = 0$), or I keep the population as it is and use it as the starting population for the new runs ($k = 100$).

Here I was able to perform a full factorial experiment of 144 combinations, each of which was targeted to take approximately 1,200 seconds. As with TS and SA, the actual CPU time varied by input settings from a mean (across the 30 runs) of 988 to 1,994 seconds (this variation is essentially due to the adaptive–N^* strategy), so as before I have calculated both raw summaries and results adjusted (via regression) for differences in CPU time.

The performance of GA, summarised in Tables 4.9–12, again varied even more dramatically according to input settings than with SA, from an adjusted mean of 0% to 66% (better than any settings of TS or SA). The conclusions in this case are clearer than with the other two algorithms.

- It is far better to use elitist strategies and at the end of every repetition to keep the population, instead of generating a new one and losing valuable time.

- The uniform and highly uniform crossover strategies are much better than the simple one-bit crossover. This and the previous conclusion help to explain why GA looked so bad in the preliminary comparison: the more recent versions of GA in the literature, with elitist strategies and more complicated crossover operations, vastly outperform the "vanilla" GA of (Holland 1975).

- Again small values of N^* (up to 10) appear best.

- Smaller values of the population size n (30 and 50) gave better results than runs with population size 80.

Table 4.9: *Results of the simulation study for GA with $p = 14$ (part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of p_{20}.*

									p_{20} (% of 20 Actual Best Models Found)		
								Mean CPU	Raw	Raw	Adjusted
r	N^*	n	p_c	p_m	c	e	k	Time (sec)	Mean	SD	Mean
22	2	80	0	0	2	1	100	1231	0.646 (0.013)	0.070	0.665
98	5	50	0	0	3	1	100	1301	0.655 (0.015)	0.083	0.664
46	2	50	0	0	2	1	100	1282	0.638 (0.019)	0.103	0.650
20	5	50	0	0	2	1	100	1307	0.630 (0.018)	0.100	0.638
16	10	30	0	0	2	1	100	1236	0.590 (0.023)	0.125	0.608
74	10	30	0	0	3	1	100	1282	0.591 (0.011)	0.058	0.603
166	2	50	0	0	3	1	100	1267	0.585 (0.014)	0.074	0.599
165	5	30	0	0	3	1	100	1335	0.588 (0.013)	0.072	0.593
89	2	80	0.3	0	1	1	100	1328	0.560 (0.018)	0.098	0.566
79	2	50	0.5	0	1	1	100	1095	0.525 (0.016)	0.087	0.561
9	5	80	0	0	2	1	100	1348	0.556 (0.016)	0.090	0.560
124	2	30	0	0	2	1	100	1453	0.565 (0.026)	0.140	0.554
158	2	80	0	0	3	1	100	1598	0.568 (0.020)	0.108	0.538
36	5	80	0.3	0	1	1	100	1342	0.531 (0.023)	0.126	0.535
237	2	30	0	0	3	1	100	1341	0.526 (0.014)	0.077	0.530
85	5	30	0	0	2	1	100	1818	0.571 (0.034)	0.187	0.513
66	5	80	0	0	3	1	100	1306	0.498 (0.024)	0.133	0.507
8	10	50	0	0	2	1	100	1399	0.501 (0.022)	0.121	0.498
35	15	30	0	0	3	1	100	1200	0.470 (0.025)	0.138	0.492
40	5	50	0	0	1	1	100	1245	0.451 (0.031)	0.171	0.468
21	2	50	0	0.01	2	0	100	1129	0.431 (0.024)	0.131	0.463
20	5	50	0.5	0.01	1	0	100	1214	0.425 (0.035)	0.191	0.446
21	5	30	0	0.01	2	0	100	1180	0.408 (0.030)	0.164	0.433
26	5	30	0.88	0.01	1	0	100	1260	0.418 (0.033)	0.179	0.433
48	2	30	0	0.01	3	0	100	988	0.380 (0.027)	0.150	0.430
21	10	30	0.88	0	1	1	100	1275	0.416 (0.027)	0.147	0.429
104	2	30	0.88	0	1	1	100	1210	0.406 (0.028)	0.154	0.428
9	15	30	0	0	2	1	100	1312	0.410 (0.026)	0.142	0.418
66	2	30	0.88	0.01	1	0	100	1111	0.383 (0.026)	0.140	0.417
37	10	50	0	0	3	1	100	1144	0.365 (0.028)	0.153	0.395
92	2	30	0	0.01	2	0	100	1534	0.411 (0.028)	0.154	0.390
27	2	50	0	0.01	3	0	100	1409	0.395 (0.023)	0.124	0.390
36	5	30	0	0.01	3	0	100	1631	0.423 (0.026)	0.142	0.389
97	2	50	0.5	0.01	1	0	100	1942	0.458 (0.026)	0.140	0.383
41	2	80	0.3	0.01	1	0	100	1301	0.373 (0.032)	0.176	0.382
81	5	30	0.88	0	1	1	100	1551	0.388 (0.030)	0.166	0.365
13	2	80	0	0.01	2	0	100	1266	0.346 (0.024)	0.129	0.360
13	2	80	0	0.01	3	0	100	1264	0.331 (0.018)	0.101	0.346
36	15	30	0.88	0	1	1	100	1290	0.320 (0.037)	0.200	0.331

Table 4.10: *Results of the simulation study for GA with $p = 14$ (part 2).*

									p_{20} (% of 20 Actual Best Models Found)		
								Mean CPU	Raw	Raw	Adjusted
r	N^*	n	p_c	p_m	c	e	k	Time (sec)	Mean	SD	Mean
11	5	50	0	0.01	3	0	100	1348	0.326 (0.022)	0.120	0.330
11	5	50	0	0.01	2	0	100	1338	0.311 (0.031)	0.171	0.316
18	5	80	0.3	0.01	1	0	100	1338	0.303 (0.029)	0.157	0.308
16	10	50	0.5	0	1	1	100	1271	0.290 (0.026)	0.145	0.303
13	10	50	0.5	0.01	1	0	100	1572	0.300 (0.036)	0.197	0.273
11	10	30	0	0.01	2	0	100	1385	0.265 (0.028)	0.153	0.263
11	10	30	0	0.01	3	0	100	1386	0.231 (0.030)	0.163	0.230
13	10	30	0.88	0.01	1	0	100	1390	0.221 (0.040)	0.220	0.219
6	5	80	0	0.01	2	0	100	1306	0.188 (0.025)	0.135	0.197
6	5	80	0	0.01	3	0	100	1282	0.173 (0.018)	0.096	0.185
15	15	50	0.5	0	1	1	100	1315	0.176 (0.031)	0.171	0.184
29	10	80	0	0	3	1	100	1429	0.171 (0.020)	0.110	0.164
10	10	80	0.3	0.01	1	0	100	1234	0.133 (0.021)	0.114	0.151
13	10	80	0.3	0	1	1	100	1294	0.138 (0.019)	0.103	0.148
5	10	50	0	0.01	2	0	100	1289	0.131 (0.021)	0.113	0.142
7	15	30	0	0.01	2	0	100	1422	0.146 (0.028)	0.155	0.140
8	15	30	0.88	0.01	1	0	100	1263	0.120 (0.027)	0.147	0.134
9	10	80	0	0	2	1	100	1419	0.135 (0.014)	0.074	0.129
21	2	30	0	0.01	2	0	0	1288	0.110 (0.014)	0.078	0.121
16	15	50	0	0	3	1	100	1116	0.075 (0.014)	0.076	0.109
7	15	30	0	0.01	3	0	100	1407	0.108 (0.016)	0.089	0.103
15	2	50	0.5	0.01	1	0	0	1162	0.066 (0.012)	0.063	0.094
20	2	30	0	0.01	3	0	0	1275	0.078 (0.010)	0.053	0.091
5	10	50	0	0.01	3	0	100	1252	0.071 (0.014)	0.079	0.087
8	2	80	0	0.01	3	0	0	1376	0.085 (0.010)	0.057	0.084
10	5	30	0	0.01	2	0	0	1296	0.073 (0.013)	0.073	0.083
4	15	50	0	0	2	1	100	1300	0.073 (0.011)	0.058	0.083
23	2	30	0.88	0.01	1	0	0	1344	0.075 (0.012)	0.066	0.078
8	2	80	0	0.01	2	0	0	1277	0.063 (0.010)	0.055	0.076
20	2	30	0	0	2	1	0	1221	0.055 (0.010)	0.056	0.075
12	2	50	0	0.01	3	0	0	1270	0.060 (0.010)	0.057	0.073
8	15	50	0.5	0.01	1	0	100	1436	0.080 (0.017)	0.091	0.073
12	2	80	0.3	0.01	1	0	0	1195	0.048 (0.010)	0.053	0.071
22	2	30	0.88	0	1	1	0	1198	0.043 (0.009)	0.048	0.066
19	2	50	0.5	0	1	1	0	1339	0.060 (0.009)	0.051	0.064
31	2	30	0	0	3	1	0	1279	0.051 (0.008)	0.046	0.064
8	2	80	0	0	2	1	0	1282	0.051 (0.009)	0.048	0.063
8	5	50	0.5	0	1	1	0	1257	0.046 (0.009)	0.050	0.062
19	2	50	0	0	3	1	0	1289	0.050 (0.010)	0.052	0.061

Table 4.11: *Results of the simulation study for GA with $p = 14$ (part 3).*

									p_{20} (% of 20 Actual Best Models Found)		
r	N^*	n	p_c	p_m	c	e	k	Mean CPU Time (sec)	Raw Mean	Raw SD	Adjusted Mean
12	2	80	0	0	3	1	0	1329	0.055 (0.010)	0.053	0.060
14	2	80	0	0	1	1	0	1295	0.050 (0.007)	0.039	0.060
3	15	50	0	0.01	2	0	100	1270	0.046 (0.010)	0.057	0.060
12	2	50	0	0.01	2	0	0	1449	0.070 (0.012)	0.068	0.060
6	15	80	0.3	0.01	1	0	100	1399	0.063 (0.014)	0.076	0.060
6	5	50	0	0.01	2	0	0	1299	0.050 (0.010)	0.052	0.059
4	5	80	0	0.01	2	0	0	1382	0.056 (0.010)	0.053	0.055
85	5	30	0.88	0.01	1	0	0	1310	0.046 (0.009)	0.047	0.055
3	10	80	0	0.01	2	0	100	1391	0.056 (0.011)	0.062	0.054
10	5	30	0	0	2	1	0	1317	0.045 (0.007)	0.040	0.052
14	5	30	0	0	3	1	0	1269	0.036 (0.008)	0.041	0.050
12	2	50	0	0	2	1	0	1375	0.048 (0.009)	0.048	0.048
11	5	30	0.88	0	1	1	0	1303	0.038 (0.008)	0.044	0.047
3	10	80	0	0.01	3	0	100	1348	0.043 (0.010)	0.052	0.046
8	5	50	0.5	0.01	1	0	0	1310	0.038 (0.008)	0.042	0.046
6	5	50	0	0.01	3	0	0	1371	0.045 (0.008)	0.042	0.045
6	5	80	0.3	0.01	1	0	0	1387	0.046 (0.008)	0.045	0.044
6	10	30	0.88	0.01	1	0	0	1342	0.040 (0.008)	0.046	0.044
5	10	30	0	0.01	3	0	0	1237	0.025 (0.007)	0.036	0.043
7	10	30	0	0	3	1	0	1238	0.025 (0.005)	0.025	0.042
6	5	80	0.3	0	1	1	0	1297	0.031 (0.007)	0.040	0.041
4	5	80	0	0	2	1	0	1398	0.043 (0.007)	0.038	0.040
9	5	30	0	0.01	3	0	0	1370	0.038 (0.010)	0.052	0.038
4	5	80	0	0.01	3	0	0	1469	0.050 (0.010)	0.055	0.037
6	5	50	0	0	2	1	0	1321	0.030 (0.007)	0.036	0.037
3	10	80	0.3	0	1	1	0	1224	0.015 (0.004)	0.023	0.034
9	5	50	0	0	3	1	0	1352	0.031 (0.006)	0.030	0.034
7	15	80	0	0	3	1	100	1327	0.028 (0.007)	0.040	0.034
2	10	80	0	0.01	2	0	0	1309	0.025 (0.006)	0.031	0.033
2	10	80	0	0.01	3	0	0	1387	0.035 (0.009)	0.047	0.033
3	15	50	0.5	0.01	1	0	0	1265	0.018 (0.007)	0.040	0.032
2	15	50	0	0.01	2	0	0	1283	0.015 (0.004)	0.023	0.026
5	15	50	0	0.01	3	0	100	1537	0.048 (0.010)	0.054	0.026
2	10	80	0	0	2	1	0	1311	0.018 (0.004)	0.024	0.026
6	10	30	0	0.01	2	0	0	1444	0.035 (0.010)	0.054	0.025
2	15	80	0	0	2	1	100	1396	0.028 (0.007)	0.038	0.025
6	5	80	0	0	3	1	0	1564	0.050 (0.010)	0.052	0.024
7	15	80	0.3	0	1	1	100	1415	0.030 (0.009)	0.051	0.024
5	10	50	0.5	0	1	1	0	1319	0.015 (0.004)	0.023	0.022

Table 4.12: *Results of the simulation study for GA with $p = 14$ (part 4).*

									p_{20} (% of 20 Actual Best Models Found)		
								Mean CPU	Raw	Raw	Adjusted
r	N^*	n	p_c	p_m	c	e	k	Time (sec)	Mean	SD	Mean
3	10	80	0.3	0.01	1	0	0	1332	0.015 (0.005)	0.026	0.020
2	15	50	0	0.01	3	0	0	1365	0.018 (0.005)	0.027	0.019
4	15	30	0.88	0.01	1	0	0	1329	0.013 (0.005)	0.029	0.019
3	10	80	0	0	3	1	0	1433	0.023 (0.006)	0.034	0.015
2	15	50	0	0	2	1	0	1308	0.006 (0.003)	0.017	0.015
4	15	30	0	0.01	2	0	0	1441	0.023 (0.007)	0.040	0.014
5	10	30	0.88	0	1	1	0	1343	0.010 (0.004)	0.024	0.014
4	10	50	0	0.01	2	0	0	1649	0.050 (0.009)	0.047	0.013
2	15	80	0	0	3	1	0	1503	0.025 (0.007)	0.036	0.008
3	15	50	0	0	3	1	0	1392	0.010 (0.007)	0.020	0.007
6	10	30	0	0	2	1	0	1471	0.020 (0.004)	0.024	0.007
4	15	30	0	0	1	1	0	1370	0.006 (0.003)	0.017	0.007
4	15	30	0	0	2	1	0	1473	0.016 (0.005)	0.027	0.003
2	15	80	0	0.01	2	0	100	1553	0.026 (0.008)	0.043	0.003
5	10	50	0	0	3	1	0	1513	0.020 (0.007)	0.036	0.001
5	10	50	0.5	0.01	1	0	0	1573	0.023 (0.006)	0.034	0.000
5	15	30	0	0	3	1	0	1465	0.006 (0.003)	0.017	0.000
4	15	30	0	0.01	3	0	0	1548	0.015 (0.005)	0.026	0.000
4	15	50	0.5	0	1	1	0	1583	0.016 (0.006)	0.033	0.000
4	10	50	0	0.01	3	0	0	1660	0.026 (0.007)	0.040	0.000
2	15	80	0	0.01	3	0	100	1634	0.023 (0.007)	0.038	0.000
4	10	50	0	0	2	1	0	1670	0.020 (0.007)	0.038	0.000
3	15	80	0.3	0.01	1	0	0	1911	0.030 (0.007)	0.038	0.000
2	15	80	0	0.01	3	0	0	1994	0.033 (0.009)	0.047	0.000
2	15	80	0	0	2	1	0	1979	0.028 (0.006)	0.033	0.000
2	15	80	0	0.01	2	0	0	1984	0.025 (0.008)	0.045	0.000
3	15	80	0.3	0	1	1	0	1889	0.011 (0.005)	0.025	0.000

Notes: (1) Negative adjusted mean p_{20} values have been truncated at 0 but appear in rank order corresponding to their untruncated values. (2) For crossover schedules 2 and 3 the p_c column is not applicable.

4.7 Comparison of optimisation methods: final results for $p = 14$

Figure 4–6 summarises all of the simulation results for the three methods in the 14–variable case. TS has the best median performance across all input settings examined, and the smallest variability of performance from worst to best. The best input settings for GA lead to the overall best performance of any method, but other choices for GA's tuning constants lead to the absolute worst results. "Vanilla" SA is in the middle, dominated by the other two methods. If the problem solved here is typical of other binary-input optimisation problems with a moderate number of $\{X_j, j = 1, \ldots, p\}$, then a brief summary of the advice arising from this study would be as follows: if you do not have a lot of time to investigate input settings in detail, then try (i) GA with an elitist strategy, keeping the whole population at the end of each iteration, a uniform or highly uniform crossover operator, and a fairly small population size (e.g., 30–50); and (ii) TS with whatever "generic" set of inputs seems reasonable to you (based perhaps on Section 4.6.1). (As mentioned above, see Chapter 6 for results with an improved version of SA.)

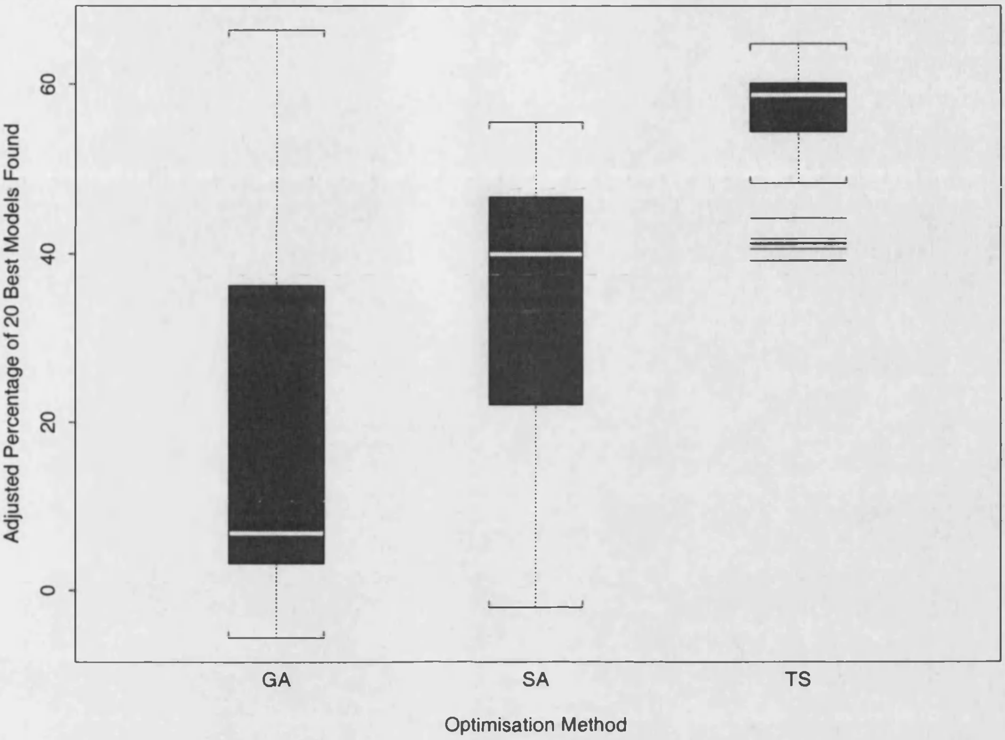
It is common practise in "vanilla" SA to take the highest point seen so far by the end of the run and apply an uphill search. I did not do this for several reasons: (1) Strictly speaking this is a hybrid of SA and local search, and for complete fairness in comparing TS, SA, and GA I decided in the main runs not to implement any hybrid strategies, so that the pure versions of each method were being compared. If I were to add a local search to the end of SA, for fairness I would need to do so also at the end of TS and GA, and one would expect this addition to have a similar effect on all three methods. (2) It would be quite difficult to decide how to choose N in the local search, and adding a local search at the end would increase the difficulty of stopping the algorithm with a fixed budget of CPU time. (3) Given the extreme multimodality of the objective function being maximised in this dissertation, I do not believe that adding a local search at the end in this way would significantly improve the results of any of the three main methods I examined.

4.8 Sensitivity analyses

4.8.1 Second full-enumeration run: different random number seed

As noted in Section 4.1, I made three full-enumeration runs with $p = 14$, the first of which was summarised in Section 4.2. In the second run I used exactly the same inputs as in the first, except the random seed, to explore the sensitivity of the findings described previously to random variation. The best model this time had 6 variables—age of patient, systolic blood pressure score, blood urea nitrogen, APACHE II coma score, shortness of breath day 1, and initial temperature—and also achieved an estimated expected utility of -7.89 ± 0.05. Figure 4–7, based on the second full-enumeration run, is analogous to Figure 4–1, and almost identical in its qualitative

Figure 4-6: *Parallel boxplots comparing the three optimisation methods in the 14-variable case.*



and quantitative conclusions: for example, the 20 best models still include the same 3 variables 18 or more times out of 20, and never include 6 other variables; and the 5 best models are minor variations on each other, and include 3–7 variables. The best model on the second run contains exactly the same variables as that on the first, with the addition of two more variables that also frequently occurred among the 20 best models in the first run.

4.8.2 Third full-enumeration run: different choice of $\left(\frac{n_M}{n}, \frac{n_V}{n}\right)$

In the third full-enumeration run I used a modelling sample of $\frac{n_M}{n} = \frac{1}{3}$ (867 patients) and a validation sample of $\frac{n_V}{n} = \frac{2}{3}$ of the data (1665 patients), rather than the other way around, to see how sensitive the results were to this aspect of the problem formulation. Figure 4–8, based on the third run, is analogous to Figure 4–1 and 4–7 and conveys virtually the same messages. The best model this time was identical to the one in the first run, although because of the change in the ratio $\frac{n_M}{n_V}$ its utility was a bit lower: -8.20 ± 0.03.

Figure 4-7: *Like Figure 4-1 but with a different random number seed.*

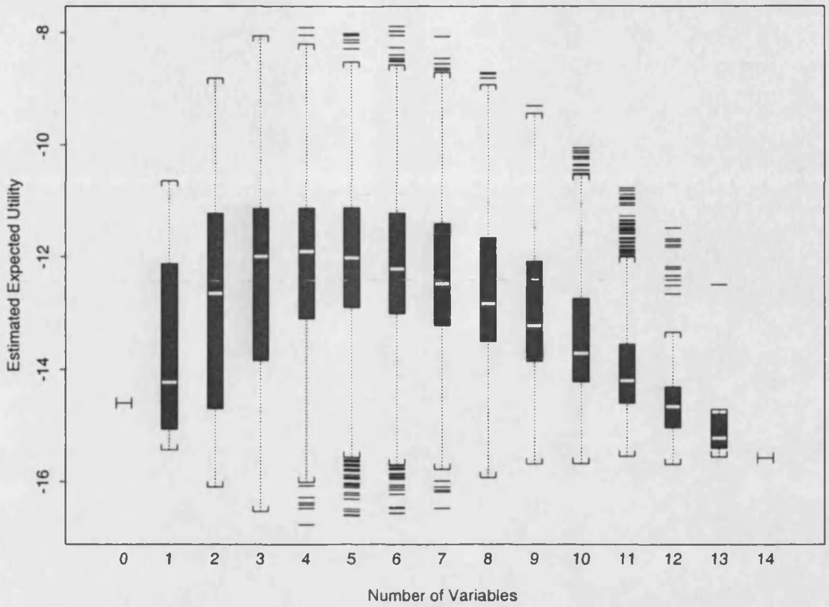
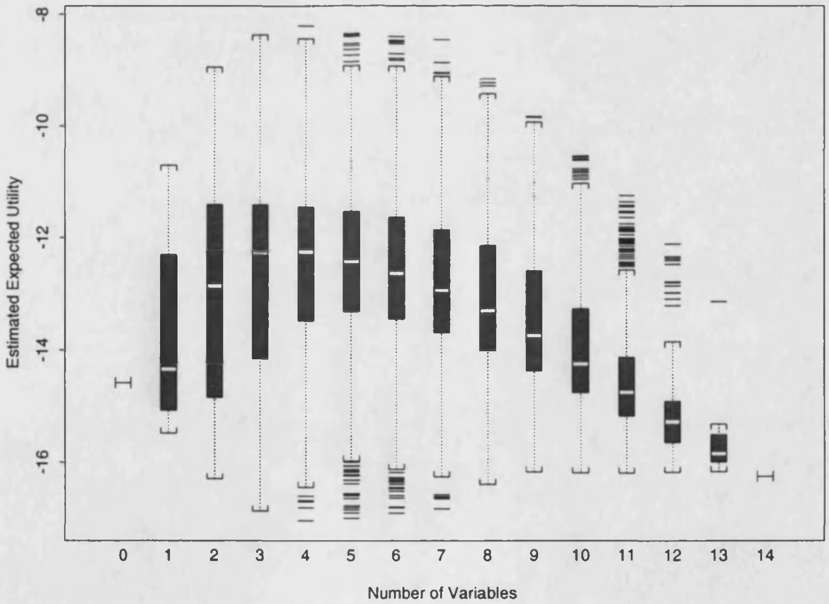


Figure 4-8: *Like Figures 4-1 and 4-7 but with* $\left(\frac{n_M}{n}, \frac{n_V}{n}\right) = \left(\frac{1}{3}, \frac{2}{3}\right)$ *instead of* $\left(\frac{2}{3}, \frac{1}{3}\right)$.



4.8.3 Quantitative comparison of the full-enumeration runs

Table 4.13 ranks the 20 best models according to full-enumeration run 1 and compares these ranks with those of the same models in runs 2 and 3. The

Table 4.13: *Comparison of the 20 best models of the first full-enumeration run with the other 2 runs.*

	Rank in Run		
Model	1	2	3
0 0 1 0 1 1 0 1 0 0 0 0 0 0	1	2	1
0 1 1 0 1 1 0 1 0 0 0 0 0 1	2	1	6
0 1 1 0 1 1 1 1 0 0 0 0 0 0	3	9	12
0 0 1 0 1 1 0 1 1 0 0 0 0 0	4	7	5
0 0 1 0 1 1 0 1 0 0 0 0 0 1	5	6	3
0 1 1 0 1 1 0 1 1 0 0 0 0 0	6	4	9
0 0 1 0 1 1 0 0 1 0 0 0 0 0	7	8	7
0 0 1 0 1 1 1 1 0 0 0 0 0 0	8	10	11
0 0 1 0 1 1 0 1 1 0 0 0 0 1	9	3	10
0 1 1 0 1 1 0 1 1 0 0 0 0 1	10	12	8
0 1 1 0 1 1 0 1 0 0 0 0 0 0	11	5	2
0 0 1 0 1 1 0 0 0 0 0 0 0 1	12	15	13
0 1 1 0 1 1 0 0 1 0 0 0 0 0	13	14	15
0 0 1 0 1 1 0 0 1 0 0 0 0 1	14	13	14
0 0 1 0 1 1 0 0 0 0 0 0 0 0	15	11	4
0 0 1 0 1 1 1 1 0 0 0 0 0 1	16	16	16
0 0 1 0 1 1 1 1 1 0 0 0 0 0	17	21	18
0 1 1 0 1 1 1 1 0 0 0 0 0 1	18	22	23
0 0 1 1 1 1 0 1 0 0 0 0 0 0	19	17	17
0 1 0 0 1 1 0 1 1 0 0 0 0 1	20	18	21

correspondence is strong but not perfect; the pairwise correlations between these three sets of ranks are 0.75, 0.85, and 0.86. To examine the correspondence in a bit more detail, I ranked all of the k–variables models (for $k = 2, \ldots, 12$) according to runs 1–3 and computed the rank correlations as a function of k, with results as in Table 4.14. The median pairwise correlation never drops below 0.84, with the minimum never falling below 0.68. I conclude that results from the first full-enumeration run are stable enough—in their definition of "truth" with $p = 14$—to use in comparing stochastic optimisation methods, as far as (a) random variation from the Monte Carlo evaluation of the estimated expected utilities and (b) choice of $\left(\frac{n_M}{n}, \frac{n_V}{n}\right)$ across the two possibilities $\left(\frac{2}{3}, \frac{1}{3}\right)$ and $\left(\frac{1}{3}, \frac{2}{3}\right)$ are concerned.

4.8.4 Penalties and rewards for prediction accuracy and marginal costs per variable

How sensitive are the optimality results to the specific choices of C_{lm}, the penalties and rewards for prediction accuracy, and c_j, the data collection marginal costs per

Table 4.14: *Pairwise rank correlations across the three full-enumeration runs as a function of the number of variables in the model.*

Number of Variables	Pairwise Correlations		
k	Min	Median	Max
2	.898	.907	.986
3	.960	.976	.980
4	.916	.925	.961
5	.800	.899	.916
6	.820	.836	.951
7	.681	.839	.841
8	.850	.896	.927
9	.804	.874	.904
10	.868	.935	.942
11	.917	.921	.968
12	.977	.981	.983

variable, given in Chapter 2? Starting with the values noted in that chapter, I multiplied all the C_{lm} by $\kappa = 2, 3, \ldots, 8$ and $\frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{8}$ (holding the data collection costs constant at their Chapter 2 values throughout) and recomputed the 20 best models in each instance in the $p = 14$ case. Results were highly stable: for instance, with $\kappa = 2$, 14 of the original 20 best models were still among the 20 best, and for 11 of the 14 variables, the frequencies of occurrence in the 20 best models differed by 10% or less. I then multiplied all the c_j by the same κ values (this time holding the penalties and rewards constant at their Chapter 2 values) and again recomputed the 20 best models. Here the findings were even more robust: for example, with $\kappa = 2$, 18 of the original 20 best were still among the new 20 best, and for all 14 variables, the frequencies of occurrence in the 20 best models differed by 10% or less uniformly. Other sensitivity analyses are of course possible; some will be described in Chapter 6.

4.8.5 Interaction terms

When constructing an admission sickness scale from available predictors X_j, it is possible to include not only main effects (the X_j themselves) but also interactions and quadratic terms (of the form $(X_j - \bar{X}_j)(X_k - \bar{X}_k)$ and $(X_j - \bar{X}_j)^2$, respectively). How sensitive are the results presented here to omission of interaction and quadratic terms among the predictors? As an approximate answer to this question, I used the entire pneumonia data set ($n = 2,532$) to find all 2-way interactions (including quadratic terms) among the 14 variables which had $z = |\hat{\beta}/\widehat{SE}(\hat{\beta})| \geq 2$ when added one by one to the 14–variable model. There were 5 such interactions out of a possible 105: between pairs of variables $(1,4), (1,5), (1,7), (1,12)$, and $(2,13)$ (using the ordering of variables in Table 4.1). I then used TS (with the best input settings from Table 4.4) with 20, 40, 80, 160, and 320 minutes of CPU time at 400MHz on the 19-variable model formed by adding the 5 new interaction terms. With a CPU time limit of 20 minutes, only one interaction appeared among the 20 best models, and for CPU time constraints in excess of 40 minutes none of the interactions appeared among the 20 best. I conclude that interactions play only a minor rôle in this problem and their omission has little effect on the findings presented here.

Chapter 5

Results in the case $p = 83$

I cannot do it without computers.

— William Shakespeare, *The Winter's Tale*

5.1 Introduction

In this chapter I describe work I have done with the full set of $p = 83$ variables. Tables 5.1 and 5.2 summarise all the predictors, together with their data collection costs and their simple correlations with 30–day death. It is evident that the original Rand 14–variable scale did a good job of choosing many of the variables which, by themselves, predict death. But which subset provides the best cost-benefit compromise?

In answering this question, the contrast with the 14–variable case could not be more extreme, because with $p = 83$ the space of all possible models is almost unimaginably large. If all 6 billion people on the planet each had a computer capable of making 10 full-enumeration runs per second (each based on $N = 500$ Monte Carlo expected utility evaluations)—which is much faster than current desktop workstations can manage—it would still take more than 5 million years to rank-order all of the $9.7 \cdot 10^{24}$ models, and even then each model's expected utility would only be known up to a Monte Carlo standard error of US\$0.05. Fortunately, this is not my goal; I wish only to identify some good models and to see how well the leading stochastic optimisation methods from Chapter 4 can find them. This chapter is a summary of work in progress; I intend to continue exploring the 83–variable case as part of the publication process.

An examination of Tables 5.1-5.2 shows that some of the variables in the $p = 83$ case that were not in the $p = 14$ case did not have very high correlation with death,

Table 5.1: *The full set of 83 variables, together with their approximate data collection costs per patient, correlation r with 30–day death, and presence in the original Rand 14–variable scale (part 1).*

Variable	Cost c_j (US$)	r	In Rand Scale?	Good? ($p = 14$)	Good? ($p = 83$)	%
systolic blood pressure score	0.17	0.29	*	**	**	73
age of patient	0.17	0.17	*	*		
blood urea nitrogen	0.50	0.32	*	**	**	100
APACHE II coma score	0.83	0.35	*	**	**	83
shortness of breath day 1	0.33	0.13	*	**	*	31
serum albumin score	0.50	0.20	*	*		
respiratory distress score	0.33	0.18	*	*		
septic complications	1.00	0.06	*			
prior respiratory failure	0.67	0.08	*			
recently hospitalised	0.67	0.14	*			
racbilateral process score	0.50	0.08				
initial temperature	0.17	−0.06	*	*	**	71
heart rate day 1	0.17	0.16			*	10
chest pain day 1	0.17	−0.15			*	31
cardiomegaly score	0.50	0.07				
plural effusion score	0.50	0.05				
pneumonia CXR score	0.67	−0.02				
ambulatory score	0.83	0.22	*			
endocarditis at admission	0.50	0.02				
CPK score	0.67	0.09				
prior antibiotics	0.17	−0.02			*	3
prior interstitial lung disease	0.17	0.02				
home oxygen use	0.33	0.10				
prior pneumonectomy	0.17	−0.02				
prior tracheostomy	0.17	−0.02				
prior aminophylline score	0.17	0.01				
hematologic history score	0.50	0.16				
cancer score	0.50	0.02				
APACHE heart rate score	0.50	0.09				
Corodaker score	0.33	−0.01				
disease of thorax	0.33	0.05				
multiple myeloma	0.17	−0.02			*	2
immunocompromised	0.17	0.01				
residence score	0.33	0.24				
hepatobiliary history	0.17	0.06				
renal history score	0.33	0.25				
APACHE respiratory rate score	0.33	0.24			*	8
new lung score	0.33	0.01				
co-morbid aspiration score	0.17	0.09			*	7
APACHE sodium score	0.67	0.14				
APACHE hematocrit score	0.50	0.10				

Table 5.2: *The full set of 83 variables (part 2).*

Variable	Cost c_j (US$)	r	In Rand Scale?	Good? ($p = 14$)	Good? ($p = 83$)	%
APACHE WBC score	0.50	0.11				
APACHE oxygenation score	0.50	0.12				
CVA score	0.33	0.14				
APACHE potassium score	0.33	0.04				
admission SBP	0.17	−0.20			**	64
CHF chest X-ray score	0.83	0.10	*			
APACHE total score	3.33	0.39	*			
respiratory rate day 1	0.17	0.22			**	92
DIA blood press day 1	0.17	0.02				
confusion day 1	0.17	0.30			*	25
pulm vasc cong score	0.17	0.07				
APACHE venus bicarb score	0.50	0.16				
pulmonary edema score	0.17	0.06				
sum of CHF components	1.83	0.11				
influenza score	0.17	−0.04			*	2
arrest in ER score	0.17	0.17			*	48
biliribin score	0.50	0.03				
positive blood culture	0.17	0.17				
positive urine culture	0.17	0.14				
wheezing at admission	0.17	−0.02				
body system count	0.83	0.33				
morbid prior copd score	0.17	−0.02				
morbid pulm hosp. score	0.17	0.03				
co-morbid cirrhosis score	0.17	0.01			*	2
co-morbid CHF score	0.17	0.08			*	33
co-morbid arrhythmias score	0.17	0.03				
co-morbid smokers score	0.17	−0.05				
co-morbid alcoholism score	0.17	−0.03			*	15
APACHE PH score	0.33	0.23				
co-morbid NGTS score	0.17	0.13				
co-morbid steroids score	0.17	0.01			*	1
sum of morbid+comorbid	2.50	0.29				
cardiac history score	0.17	0.06				
neurologic history score	0.17	0.28			*	1
oncologic history score	0.17	0.02				
immunologic history score	0.17	0.01				
musculoskeletal score	0.17	0.17			*	5
APACHE temperature score	0.33	0.02				
APACHE mean BP score	0.33	0.08				
APACHE creatinine score	0.33	0.20				
DX score	0.33	0.07				
sex of the patient	0.17	0.02				

Note: The final three columns of the table are explained in the text below.

suggesting that the dimensionality of the space worth searching was less than that implied by including all 83 predictors. However, (a) many of the new variables did in fact have quite a high marginal correlation with death (in fact, higher than some of the variables in the final 14-variable Rand scale), and (b) with $p = 83$ I wanted to learn about how well the three optimisation methods performed in a very high-dimensional space and without a great deal of "coaching" about the value of the possible inputs. In Chapter 6 I explore the "value added" of improving SA's choice of which variables to bring in and take out of the current working model by using a measure that takes correlation with death into account; see Section 6.1 for results of this approach both with $p = 14$ and $p = 83$.

Since full enumeration is impossible with $p = 83$, the first task was to create a workable proxy for it. To create such a proxy I did the following.

- First I chose one "good" input configuration each from tabu search (TS), simulated annealing (SA), and the genetic algorithm (GA) (from Tables 4.4, 4.6, and 4.9), where "good" means a compromise between the best results from $p = 14$ and a desire for each method to visit a lot of models. In practice I chose an input configuration for each method that was among the top 15 with $p = 14$; and

- Then I ran each algorithm with these "good" configurations for one week of CPU time at 400 MHz on the $p = 83$ case (using random starting models). Each method visited about 630,000 models in that time; the total across the three methods was 1,900,377, although this figure included a lot of duplicate models. (Everything about the 83–variable case was ponderous: the resulting file required 1.8 gigabytes of disk storage, and processing it in `Stata` required 1 gigabyte of virtual memory.)

- I eliminated all of the duplicates, arriving finally at 825,635 unique models visited by the three methods in one week each. I then sorted these models on their apparent utility (in each case N was either 1 or 4 or 5, based on the adaptive method), and extracted the 3,000 best models on this basis. Finally I then ran the full-enumeration program on these 3,000 models (with $N = 500$) to find their "real", as opposed to apparent, expected utility, and sorted them one more time on their real utility values.

In this chapter the 3,000 models, and their utilities obtained in this way, will be regarded as "truth" for the purpose of comparing GA, SA, and TS with $p = 83$.

A full experiment comparing these three methods, like that in Section 4.6 in the 14–variable, case would have taken far too long, for two reasons: (a) with 83 variables a single evaluation of the estimated expected utility took 1.3 seconds at 400 Unix MHz, 3.3 times longer than with $p = 14$, and (b) 20 minutes of CPU time was far from sufficient for any of the methods to begin finding good models. So instead I performed an abbreviated experiment guided by the results in the 14–variable case. I chose the 10 best input settings for each method from the tables in Sections 4.6.1–4.6.3, and gave each algorithm 3 hours of CPU time at each setting, repeating this process 5 times in each case and averaging the results to diminish random variation in the findings. To get a preliminary idea of how much better the optimisation methods could do with a larger CPU budget, I also made one run with each method, using the highest-performance settings with $p = 14$ and allowing the methods 24 hours of CPU time in each case. In the remaining sections of this chapter I summarise the results of these experiments.

5.2 One-week results

Table 5.3 gives the input settings used by the three optimisation methods in the one-week runs. Because the adaptive–N^* method was in use, the number of modelling/validation splits on which each model's estimated quality was based varied from 1 to 5; in the end the distribution of the actual N used with the 3,000 best models found (on the basis of apparent utility) was $(69, 31)\%$ across the values $N = (1, 5)$. 81% of the 3,000 best models (Table 5.4) were found by GA, with 19% discovered by TS and only 1 out of the entire 3,000 (0.03%) obtained by SA. When I restricted attention only to those models with $N > 1$, for which the utility determination was more accurate, the results were even more striking in favour of GA: 95% from GA, 5% from TS, 0.1% from SA. This is the first indication that GA may overwhelmingly be the best method with $p = 83$.

Figure 5–1 summarises the estimated (real) expected utility from the 3,000 best models found in the one-week runs, as a function of the number of variables in the model. This plot is a rough analogue of Figures 4–1, 4–7, and 4–8, with the roughness appearing because this is not a full enumeration of all models with 1–22 variables. Even so, the approximately quadratic shape traced out by the medians and maxima of most of the boxplots is clear, and demonstrates that in the 83–variable case the best models have 5–10 variables. This is only slightly larger than with $p = 14$, where the optimal range was 4–7, even though the optimisation methods have 69 more variables to work with; this is because the Rand scale harvested so many of

Table 5.3: *Input settings for the one-week runs.*

Tabu Search		Simulated Annealing	
Input	Value	Input	Value
N^*	5	N^*	4
# of Preliminary Searches	9	Initial Temperature	1.0
# of Intensification Searches	10	Final Temperature	0.1
Maximum # of Restarts Within Each Intensification	3	Temperature Schedule	Reciprocal
# of Diversification Searches	14		

Genetic Algorithm	
Input	Value
N^*	5
Population Size	50
Crossover Probability	—
Mutation Probability	0
Crossover Strategy	Highly Uniform
Elitist Strategy?	Yes
Percentage of Population Retained	100%

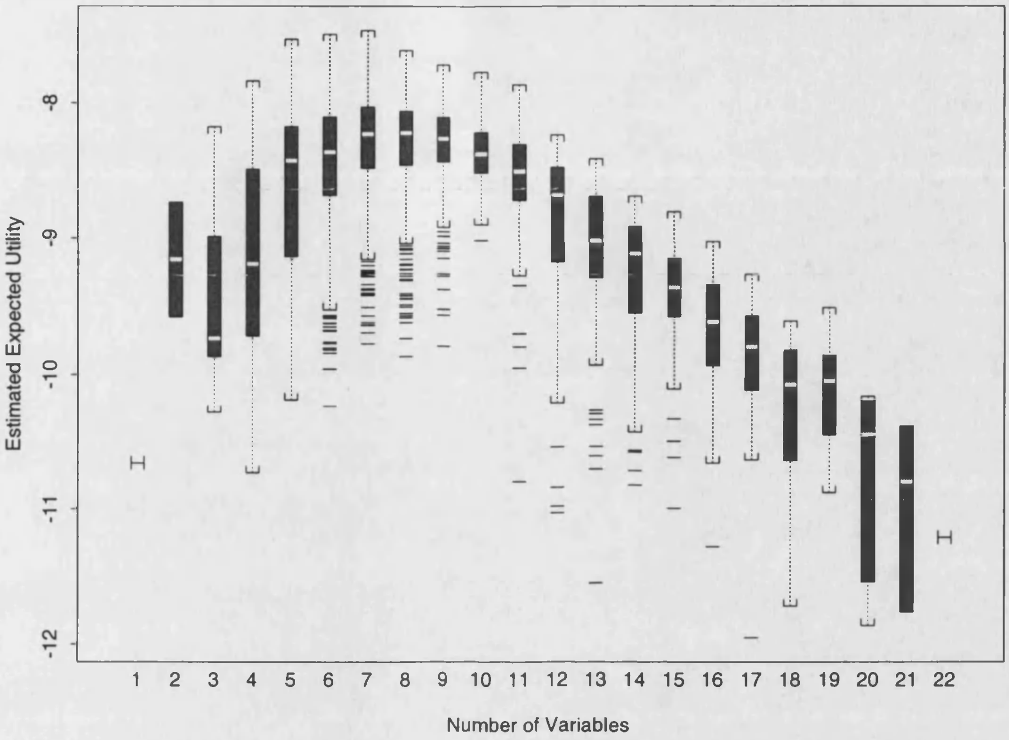
Table 5.4: *Distribution of model dimension in the 3,000 best models from the one-week runs, by optimisation method.*

	All N			$N > 1$		
Method	Mean	SD	%	Mean	SD	%
GA	7.5	1.7	81.20	7.3	1.4	94.9
SA	18.0	0.0	0.03	18.0	0.0	0.1
TS	14.9	2.0	18.77	12.5	1.2	5.0
Total	8.9	3.4	100.0	7.6	1.8	100.0

the variables with good univariate predictive performance. Table 5.4 summarises the dimensions of the models found by the three methods; it is clear from this table that GA achieves its good results by finding its way faster (from a random starting point) to the smaller models where the best utilities are concentrated.

In a manner analogous to the situation with $p = 14$, where policy-makers might well wish to look among the (say) 20 best models for the one finally chosen in the league-table quality analysis, in the 83–variable case a larger value like the 100 best models might be useful. Table 5.5 presents an overall summary of the 3,000 best and

Figure 5-1: *Estimated (real) expected utility as a function of number of predictors retained, based on the 3,000 best models found from the one-week runs with* $p = 83$.



the 100 best models found in the one-week runs, including the "real" utilities (with Monte Carlo standard error US\$0.05) from the full-enumeration run with $N = 500$ (all of the 100 best models were found with GA). The gap between real and apparent utilities (the latter being based typically only on about $N = 5$) is more than US\$1 on average across the 100 best models, demonstrating the value of conducting a full-enumeration exercise once the optimisation methods have found a number of candidate models.

Columns 6 and 7 of Table 5.1 identify the most promising variables from the one-week runs. The second of these columns gives the percentage of time each variable appeared among the 100 best models if that frequency was at least 1%. 61 of the 83 variables fail this test. The six most common variables in the 100 best models, denoted with double asterisks in column 6, were blood urea nitrogen, respiratory rate day 1, APACHE II coma score, systolic blood pressure score, initial temperature, and admission systolic blood pressure. Four of these variables were identified in the parallel exercise with $p = 14$, but two are new: it would appear that there is extra information in the actual values of the admission systolic blood pressure and respiratory rate on day 1, above and beyond what is present in the similar scales

Table 5.5: *Summary of the 3,000 best and the 100 best models found in the one-week runs.*

	3,000 Best				100 Best			
Summary	Mean	SD	Min	Max	Mean	SD	Min	Max
N	2.2	1.9	1	5	4.4	1.4	1	5
Dimension	8.9	3.4	1	22	7.1	1.1	5	10
Apparent Utility	−7.14	0.33	−7.48	−5.44	−6.67	0.51	−7.48	−5.44
Real Utility	−8.60	0.65	−11.95	−7.47	−7.75	0.07	−7.82	−7.47

among the 14 Rand variables.

The overall best model found in the one-week runs had 7 variables: systolic blood pressure score, blood urea nitrogen, APACHE II coma score, initial temperature, admission SBP, respiratory rate day 1, and arrest in ER score. The real utility achieved by this model, −7.47, is only US$0.43 better than the corresponding figure with $p = 14$; this is again a consequence of the Rand 14-variable scale being so heavily packed with variables with good univariate predictive behaviour.

5.3 A simulation experiment with $p = 83$

As mentioned in Section 5.1, in the 83–variable case I carried out a smaller experiment (due to computer time constraints) exploring the performance of TS, SA, and GA as a function of input settings. I chose small variations on the 10 best settings for each method with $p = 14$, with the aim of producing runs that would take 3 hours of CPU time each, and I repeated each of these runs 5 times with different random number seeds and averaged the results. Several of the methods performed so poorly that there was no point in using the percentage of the 100 (or 3,000) best models found as the main outcome; I focused instead on the following three performance summaries.

- The dimensions of the models visited. Tables 5.4–5 and Figure 5–1 show that this is a good proxy for the quality of the models: if a method only finds models with (say) 15–25 variables then it has not found good models;

- The apparent utilities of the 100 best models visited. It would have taken too long to evaluate the real utilities of all of these models; to give a flavour of what those results would have been I merged the 5 runs with any given input settings and method, extracted the 10 best according to their apparent utilities, and did full-enumeration on these 10 best; and

- A measure of the *efficiency* of each method in finding as many of the 3,000 best models as possible. This score will be explained below.

I focused both on all models visited during the 3–hour runs and on the 100 best models found in each of these runs.

To motivate the efficiency measure I used, suppose that we are only interested in the top five models, and we wish to summarise the quality of the subset of these models found by a given method. Imagine that one method found models $(1, 2, 4)$, and another method found models $(2, 3, 4, 5)$. Which of the two methods is better? To decide this, I attach a linear weight to each of the top five models, so that the best model has weight 5 and the worst 1. Then a *deficiency* score may be computed by summing the weights of the five best models that were not found by the method. So method one above would have a deficiency of 4 (since models 3 and 5 are missing, which corresponds to weights 3 and 1 respectively), and method two would have a deficiency of 5 (since only model 1 is missing). So according to this measure the run that found models (1, 2, 4) is slightly better. Instead of monitoring the deficiency I report the *efficiency* of the method, which is the sum of all the numbers from 1 to 5 in the above example, minus the deficiency. Thus method one that found models (1, 2, 4) has an efficiency score of 11, while method two that found models (2, 3, 4, 5) has an efficiency score of 10, out of a possible $15 = \sum_{i=1}^{5} i$. In the simulation experiment, each run with a given input setting visited a random number of models, typically on the order of 1,000–10,000; for each run the efficiency score was computed across all models visited.

As was true in the 14–variable case, the actual CPU times of the runs fluctuated around their target value of $10,800$ seconds (3 hours) at 400 Unix MHz. Because of computing limitations I was not able to make enough runs to serve as the basis of a reliable adjustment for CPU variations, so I present unadjusted findings in the tables. Even with some variation in CPU time, clear patterns emerge as to which methods (and which input settings) perform best.

In what follows it will be seen that GA ended up contributing most of the good models to the proxy "truth", which might appear to provide a favoured status for GA when the methods are compared. I do not believe that this interpretation is justified, for the following two reasons: (1) The proxy was created by giving all three methods—GA, SA, and TS—an equal chance to contribute good models to the eventual listing of the best models, because all three methods had the same amount of CPU time (1 week). (2) The only way in which the proxy appeared in my results with $p = 83$ was in the calculation of the efficiency measure; an appreciation of the performance of the methods which has nothing at all to do with the proxy may be gained by looking at the mean real utility rows in Tables 5.6-5.8.

5.3.1 Tabu search

Table 5.6 summarises the input settings and results from the experimental runs on TS with $p = 83$ (naming conventions for inputs are as in Section 4.6.1; the rows with a single (double) asterisk refer to the 100 (10) best models found). In these runs N^* varied from 5 to 20, the total number of runs from 1 to 4, the number of preliminary searches from 2 to 14, the number of intensification searches from 2 to 15, the maximum number of restarts within the intensification search from 0 to 6, and the number of diversification searches from 1 to 4. The mean CPU times of the runs ranged from 10,044 to 15,361 seconds, with the total number of models visited during the approximately 3 hours of CPU time varying from 686 to 2,330.

Most of the input settings examined resulted in searches that concentrated on models with too many variables to have good performance—the mean model dimension in the 100 best models ranged from 9.5 to 29.7, with most of the minimum dimensions above 10 and many of the maximum dimensions above 20. Input settings 3 and 4 gave the best results, yielding real utilities for the 10 best models of about US\$−8.60 and routinely finding about 4 of the 6 best variables from the full-enumeration runs, but none of the 50 TS runs managed an efficiency score greater than 0, i.e., not a single model among the 3,000 best was ever visited. Three hours of CPU time does not appear to be enough for TS to get anywhere near the global optimum (although this does not mean that 3 hours was a bad choice for the experiment; as will become clear below, GA's performance with 3 hours of CPU time is already rather good).

5.3.2 Simulated annealing

Table 5.7 gives the input configurations and results for each of the 10 runs I made using SA with $p = 83$ (naming conventions for the inputs are as in Section 4.6.2). In these runs N^* varied from 3 to 10, the initial and final temperatures ranged from 0.5 to 10.0 and 0.05 to 0.1, respectively, and I examined all four temperature schedules. The mean CPU times actually observed varied from 8,629 to 12,947 seconds. SA typically visited a lot more models than TS—its total number of models ranged from 1,434 to 8,762—but in most cases the great majority of these models were examined with $N = 1$, indicating that SA was having a hard time finding good models.

The typical dimensions of the models visited by SA were even larger than with TS: the mean dimension across the 100 best models varied across the input settings

Table 5.6: *Input settings and results for the 3-hour runs of TS in the simulation experiment with* $p = 83$ *(SDs in parenthesis).*

	Run									
Input	1	2	3	4	5	6	7	8	9	10
r	1	3	2	4	2	1	2	3	3	2
N^*	15	7	7	9	8	15	20	9	5	11
l	10	2	2	2	14	9	11	2	9	2
i	6	7	4	4	15	12	2	4	3	4
t	3	2	6	1	5	3	0	2	3	1
d	3	2	4	2	1	3	1	2	2	3

	Run									
Result	1	2	3	4	5	6	7	8	9	10
Mean CPU (10K sec)	1.00	1.07	1.54	1.33	1.20	1.04	1.39	1.44	1.34	1.03
# ($N = 1$)	74	233	134	340	664	57	272	357	403	211
# ($N > 1$)	645	1467	2196	1627	1350	629	598	1476	1828	941
Mean* Dimension	28.0 (3.1)	13.9 (3.0)	12.7 (2.0)	9.5 (1.9)	18.5 (3.4)	29.7 (5.4)	27.3 (3.3)	14.3 (3.9)	14.6 (2.1)	18.9 (3.0)
Min* Dimension	26.6	10.4	10.2	6.2	13.4	27.8	24.6	11.0	9.8	16.4
Max* Dimension	29.8	18.8	16.4	15.6	24.0	32.2	34.2	18.0	20.8	22.8
Mean* Apparent Utility	−15.0	−10.1	−9.5	−9.4	−11.6	−15.4	−14.9	−10.7	−10.5	−11.8
Mean** Real Utility	−13.0	−9.4	−8.6	−8.6	−10.4	−13.1	−12.6	−9.8	−9.7	−10.2
Mean** Dimension	23.1 (0.3)	10.9 (1.2)	9.3 (0.9)	6.8 (1.4)	12.6 (4.2)	23.1 (0.3)	22.4 (0.8)	11.5 (1.6)	10.8 (3.7)	14.7 (1.4)
# 6** Best	1	3	4	4	2	1	2	2	2	2
Mean Efficiency	0	0	0	0	0	0	0	0	0	0

Notes: (1) # ($N = 1$) is the number of models visited with $N = 1$, and analogously for # ($N > 1$); # 6 Best is the number of the variables marked with two asterisks in column 6 of Table 5.1 which occurred at least 50% of the time in the 10 best runs. (2) Rows marked with one (or two) asterisks refer to the 100 (or 10) best models found.

from 15.3 to 25.0, the minimum dimension never dropped (on average) below 12.4, and the maximum was frequently above 25. The mean apparent utility in the 100 best models fell below −10 for seven of the 10 input settings, and the mean real utility in the 10 best models found was below −9 with eight of the 10 settings. SA

Table 5.7: *Input settings and results for the 3-hour runs of SA in the simulation experiment with $p = 83$ (SDs in parenthesis).*

Input	Run 1	2	3	4	5	6	7	8	9	10
N^*	4	5	5	5	3	10	10	5	4	10
T_0	1.0	0.5	0.5	2.5	1.0	0.5	1.0	1.0	1.0	10.0
T_f	0.1	0.05	0.05	0.1	0.1	0.05	0.1	0.1	0.1	0.1
Cooling Schedule	3	1	2	3	3	4	4	3	2	3

Result	Run 1	2	3	4	5	6	7	8	9	10
Mean CPU (10K sec)	1.23	0.86	1.22	1.01	0.97	0.91	1.29	0.86	1.26	0.92
# ($N = 1$)	7663	5326	6230	3736	5901	182	1062	3993	6813	1170
# ($N > 1$)	1099	533	1093	1285	941	1252	1459	912	1149	818
Mean* Dimension	22.4 (5.4)	23.5 (1.1)	22.1 (2.4)	18.6 (3.9)	22.6 (4.4)	16.4 (3.5)	15.3 (4.6)	19.5 (3.5)	25.0 (2.7)	15.5 (3.6)
Min* Dimension	17.6	17.8	18.4	15.6	17.8	14.0	12.8	15.6	19.8	12.4
Max* Dimension	28.2	31.4	27.8	23.0	29.4	19.2	18.2	25.8	31.0	19.6
Mean* Apparent Utility	−11.0	−11.8	−11.1	−10.3	−11.3	−9.7	−9.2	−10.4	−11.9	−9.9
Mean** Real Utility	−9.5	−11.3	−10.4	−9.0	−9.6	−9.0	−8.2	−9.2	−10.5	−8.8
Mean** Dimension	14.8 (1.5)	20.3 (1.3)	18.8 (0.8)	13.0 (0.9)	14.0 (2.7)	10.6 (1.3)	9.9 (0.9)	13.6 (1.1)	20.4 (1.0)	11.1 (1.1)
# 6** Best	1	2	2	3	2	3	3	1	5	4
Mean Efficiency	0	0	0	0	0	0	0	0	0	0

Notes: See Table 5.6.

typically only found 1–3 of the six best variables among its 10 best models, and (as was true for TS) never achieved an efficiency of discovering any of the 3,000 best models above 0. Input setting 6 was the best, but overall SA's performance with $p = 83$ is, if anything, even worse than in the 14–variable case. (However, in Chapter 6 I report results of an improved SA method that are much better.)

5.3.3 Genetic algorithm

Table 5.8 presents the input settings and results from the simulation study for GA with $p = 83$ (naming conventions for the inputs are as in Section 4.6.3). In the runs reported on here N^* varied from 2 to 10, I used three different crossover operators (one-way or simple, uniform, and highly uniform), the elitist strategy was used in all cases (meaning that there was no mutation), the population was retained at the start of each new repetition in all runs, and the population size ranged from 30 to 80. Mean CPU times varied from 9,590 to 14,485 seconds. GA visited on average between 2,959 and 12,470 models as a function of input settings. The mean ratio of visits with $N = 1$ to those with $N > 1$ is consistently greater than 1; this was a marker of poor performance with SA, but interestingly GA, as the rest of the table shows, achieves good results even so.

Input settings 9 and 10—the only ones with the one-way crossover operator—performed much worse than the other eight settings examined; I will say no more about them. Apart from this, the dimensions of the models visited by GA are much closer to the region of good performance as indicated by Figure 5–1: the mean dimensions of the best 100 models with the best input settings for GA were consistently below 10, and the minima were frequently below 6. This translates into comparatively excellent results for the apparent utility of the 100 best models (often below −8) and the real utility of the 10 best (never much greater than −8). The 10 best models typically had between 7 and 9 variables on average, and most of the six best predictors from Table 5.1 were frequently located.

GA is the first method to achieve non-zero efficiencies: in absolute terms the proportions of the best 3,000 models found are not stunning (the maximum possible efficiency score is about 4.5 million), and the standard deviations show that there is considerable random variation in efficiency achieved (every input setting had at least one run with efficiency 0), but several of the input settings with GA managed to attain efficiencies of 40,000 to 235,000 on average with 3 hours of CPU time, and one run attained an efficiency (732,623) equivalent to having found the 254 top models among the 3,000 best. Input setting 8—with $N^* = 5$, a population size of 30, elitist and highly uniform crossover strategies, and 100% retention of the current population at the start of each new repetition—is both the clear GA favourite and the overall winner across all methods, although GA settings 1 and 6 are also promising.

Table 5.8: *Input settings and results for the 3-hour runs of GA in the simulation experiment with $p = 83$ (SDs in parenthesis; for explanation of other symbols see Table 5.6).*

	Run									
Input	1	2	3	4	5	6	7	8	9	10
N^*	2	5	2	5	10	10	2	5	2	2
n	80	50	50	50	30	30	50	30	80	50
p_c	—	—	—	—	—	—	—	—	0.3	0.5
p_m	0	0	0	0	0	0	0	0	0	0
Crossover Strategy	2	3	2	2	2	3	3	3	1	1
Elitist Strategy?	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
k	100	100	100	100	100	100	100	100	100	100

	Run									
Result	1	2	3	4	5	6	7	8	9	10
Mean CPU (10K sec)	1.06	1.11	1.17	1.10	1.15	1.13	0.96	1.35	1.45	1.07
# $(N = 1)$	6622	3963	9449	3343	1664	1920	7826	7210	9343	7178
# $(N > 1)$	2926	1957	2586	2065	1295	1285	1958	2265	3127	2114
Mean* Dimension	9.6 (1.4)	9.8 (0.6)	9.7 (0.8)	9.9 (0.4)	11.1 (2.1)	8.6 (1.0)	11.4 (0.8)	8.7 (1.4)	19.0 (2.4)	21.3 (1.1)
Min* Dimension	6.2	5.2	6.4	6.8	8.0	5.4	6.4	5.4	15.8	17.6
Max* Dimension	13.8	14.8	12.8	13.4	14.6	12.2	17.4	12.2	23.8	26.0
Mean* Apparent Utility	−6.9	−7.8	−6.6	−7.5	−8.3	−7.7	−7.6	−7.2	−9.9	−10.6
Mean** Real Utility	−8.1	−8.3	−8.1	−8.2	−8.1	−8.0	−8.6	−7.8	−10.6	−10.8
Mean** Dimension	7.9 (1.9)	9.0 (1.3)	8.5 (1.1)	9.1 (1.1)	7.5 (0.8)	7.6 (1.4)	8.8 (2.4)	7.1 (1.3)	16.0 (1.2)	19.2 (1.2)
# 6** Best	4	4	6	4	2	4	3	5	6	4
Mean Efficiency (1K)	57.7	10.0	10.2	1.1	0.3	37.9	0.2	234.7	0	0
SD Efficiency (1K)	95.2	15.9	18.9	2.5	0.8	64.6	4.3	299.4	0	0
Max Efficiency (1K)	225.1	37.7	43.7	5.6	1.7	149.9	9.7	732.6	0	0

Table 5.9: *Input settings and results for the 24-hour run of TS in the simulation experiment with $p = 83$.*

Input	r	N^*	l	i	t	d	Result	CPU Time	Number of Models Visited With $N = 1$	Number of Models Visited With $N > 1$
Value	27	5	10	6	3	3	Value	86,402	37,235	5,653

	100 Best Models							
	Dimension				Apparent Utility			
Result	Mean	SD	Min	Max	Mean	SD	Min	Max
Value	9.54	1.49	6	13	−7.80	0.131	−7.89	−7.19

	10 Best Models				
	Real Utility		Dimension		
Result	Mean	SD	Mean	SD	# 6 Best
Value	−8.53	0.199	8.20	1.03	2

5.4 Results with 24 hours of CPU time

It is interesting to consider how much better the optimisation methods would do with a larger budget of CPU time than 3 hours. To obtain preliminary information along these lines, I chose the best input configuration for each of TS, SA, and GA from the $p = 14$ results and made one run with 24 hours of CPU time for each of these input settings. In a manner analogous to the results in the previous section I looked at results both for all models visited and for the 100 best models found (on the basis of apparent utility), and I also extracted the 10 best models found by each method and ran the full-enumeration program to obtain their real utilities. The actual CPU times for all three methods matched almost perfectly the target of 24 hours (86,400 seconds) because, after estimating values of the input parameters so that the runs would finish at around 24 hours I simply halted the C programs when that amount of CPU time was reached.

5.4.1 Tabu search

Table 5.9 summarises the input settings and results from the 24-hour run with TS in the 83-variable case. The input configuration closely resembles, but does not match exactly, input setting 9 from Table 5.6 (except of course for the number r of repetitions of the whole algorithm, which is 9 times larger with 24 hours than with 3). The performance of TS is naturally much better with 24 hours of CPU time: across the 100 best models the mean dimension has dropped to 9.5 (minimum

Table 5.10: *Input settings and results for the 24-hour run of SA in the simulation experiment with* $p = 83$.

Input	N^*	T_0	T_f	Cooling Schedule
Value	4	1.0	0.1	3

Result	CPU Time	Number of Models Visited With $N = 1$	Number of Models Visited With $N > 1$
Value	86,404	62,912	1,004

	100 Best Models							
	Dimension				Apparent Utility			
Result	Mean	SD	Min	Max	Mean	SD	Min	Max
Value	20.4	3.39	17	32	−10.2	0.346	−10.5	-9.10

	10 Best Models				
	Real Utility		Dimension		
Result	Mean	SD	Mean	SD	# 6 Best
Value	−10.6	0.452	18.2	1.13	5

6, maximum 13) and the mean apparent utility has fallen to −7.8, and among the 10 best the mean real utility has declined to −8.5 and the mean dimension to 8.2. Unfortunately it is still true that the efficiency of this run was 0, so TS has still not made its way to the best models (with this input configuration, at least).

5.4.2 Simulated annealing

Table 5.10 gives the input settings and results for the 24–hour run with SA, which was identical to configuration 1 in the 3–hour runs. A comparison with the corresponding column of Table 5.7 reveals that SA has managed little or no improvement on its 3–hour performance, even with a budget of CPU time eight times the previous size, and (because of random variation) has actually appeared to go backwards in some respects. The 24–hour SA run does look better than its 3–hour counterpart in the number of the six high-frequency good variables (Table 5.1) it has found, but this is only because it is fixated on models with far too many predictors. Like TS, simulated annealing failed to find a single one of the 3,000 best models from the one-week full-enumeration runs in 24 hours, but in all other important respects the 24–hour TS run uniformly dominated that of SA.

Table 5.11: *Input settings and results for the 24-hour run of GA in the simulation experiment with* $p = 83$.

Input	N^*	n	p_c	p_m	c	e	k
Value	2	80	—	0	2	1	100

Result	CPU Time	Number of Models Visited With $N = 1$	Number of Models Visited With $N > 1$
Value	86,402	104,204	10,706

	100 Best Models							
	Dimension				Apparent Utility			
Result	Mean	SD	Min	Max	Mean	SD	Min	Max
Value	8.47	0.936	7	11	−5.66	0.256	−5.92	−4.64

	10 Best Models				
	Real Utility		Dimension		
Result	Mean	SD	Mean	SD	# 6 Best
Value	−7.91	0.160	8.40	1.07	2

5.4.3 Genetic algorithm

Table 5.11 provides the input settings and some summaries from the 24–hour run with GA, which again used inputs that were identical to the first input configuration for GA in the 3–hour runs. Unlike SA, GA has made good use of the extra CPU time: for example, in comparison to the mean of the corresponding 3–hour runs, the mean dimension of the 100 best models is 12% lower, and the attained real utility (−7.91) is not far from the global optimum of −7.47 found with the full enumeration based on the one-week runs. This particular 24–hour GA run managed to find 9 of the best 3,000 models, ranging in rank from 71 to 2,222, yielding an efficiency score of 17,127. (This is lower than the mean of the 3–hour runs due to random variation; the same reason explains why only 2 of the 6 best variables appeared with at least 50% frequency across all models visited.) Once again GA dominates both of the other two methods across the 24–hour runs, although TS has closed the gap.

5.5 Comparison of optimisation methods: final results with $p = 83$

- GA is the clear winner in the 83–variable case with both 3 and 24 hours of CPU time at its disposal (but see the results for $p = 83$ in Section 6.2.1 on the performance of the improved version of simulated annealing). The best input

Table 5.12: *Comparison of the total number of utility evaluations achieved by the three optimisation methods in the 24-hour runs.*

Method	N^*	Number of Models Visited With $N = 1$	Number of Models Visited With $N > 1$	Number of Utility Evaluations
TS	5	37,235	5,653	65,500
SA	4	62,912	1,004	66,928
GA	2	104,204	10,706	125,616

settings for the 83–variable case I have found so far are as follows: $N^* = 5$, population size 30, highly uniform crossover strategy, elitist selection (and therefore no mutation), and retention of 100% of the current members of the population at the beginning of each repetition of the algorithm. These settings are similar to those that came in second (out of 144 combinations examined) in the $p = 14$ case, so they appear to be robust to the dimension of the problem, at least in the range from fairly small to fairly large.

- TS came in second with $p = 83$, which is something of a contrast with the 14–variable case, where the best implementations of the two methods did about equally well. Of the input configurations I examined for TS with 83 variables the best was the following: 4 overall repetitions of the algorithm, $N^* = 9$, 2 preliminary searches, 4 intensification searches, 1 maximum restart within the intensification process, and 2 diversification searches.

- Vanilla SA again came a poor third in the 83–variable case, with the amount by which it trailed the other two methods growing as p increases. (As mentioned above, see Chapter 6 for results with an improved version of SA.)

- It is interesting to speculate about the reason for GA's dominance. Table 5.12 compares the three optimisation methods in the number of utility evaluations each is able to achieve in the 24–hour runs. (Similar results are evident with the 3–hour runs.) In the same amount of CPU time, and bearing in mind that the programs for all three methods were written with an attempt at equal efficiently in the same language, GA is able to find the time to evaluate almost twice as many utilities as the other two methods. I believe that this difference is attributable to the amount of extra "overhead" required by TS and SA that is not present in GA: SA spends a noticeable amount of time making calculations (involving expensive calls to the logarithm and exponential functions) to support its cooling schedule and acceptance probabilities, and TS uses a fair amount of CPU time managing the tabu list.

Chapter 6

Conclusions and extensions

> Q: How long should a [person's] legs be?
>
> A: Long enough to reach the ground.
>
> — Abraham Lincoln

6.1 Summary of the project and its main findings

In this dissertation I used data from a 5–year study conducted by the Rand Corporation in the 1980s, of approximately 2,532 people in the US, to construct cost-effective scales measuring the sickness-at-admission of elderly patients hospitalised with pneumonia (Chapter 1). Starting with a model chosen by Rand with 14 sickness variables, my aim was to find a smaller model that achieves a better compromise between predictive accuracy and data collection costs. I began with a problem formulation proposed by (Draper 1996), in terms of maximisation of expected utility, to achieve this compromise in a way that is relevant to health policy (Chapter 2). The space of all possible models in the 14–variable case is small enough to permit a full enumeration of accurate estimates of the expected utility of all $2^{14} = 16,384$ subsets of the predictors. These estimates are based on a Monte Carlo evaluation of the expected utility using averages of N random splits of the available data into modelling and validation subsamples. By conducting the full enumeration in the 14–variable case (with $N = 500$) I showed that, with realistic costs, the optimal solution, if implemented in a league-table or input-output approach to quality assessment on a wide scale, could result in significant monetary savings.

The main problem, however, was that there were not just 14 sickness variables originally available for pneumonia—there were $p = 83$, and 2^{83} is such a large number of models that full enumeration fails. It was clear from the 14–variable case that the expected utility function has many local optima. A global optimisation method is needed, and it has been shown that stochastic optimisation is a promising option in the presence of a large number of local solutions. So I conducted a literature review of more than 100 articles from the stochastic optimisation literature (Chapter 3), and identified five approaches for further study: genetic algorithms (GA), messy simulated annealing (MSA), simulated annealing (SA), tabu search (TS), and threshold acceptance (TA). The general problem I addressed from an optimisation point of view is that of maximising a real-valued function of a binary input vector of length p.

After exploring the geometry of the solution space and the optimal choice of N (Chapter 4), I performed a preliminary simulation experiment with the five optimisation methods listed above in the $p = 14$ case, using the full-enumeration results as truth against which the methods could be compared. It was clear that MSA and TA were either dominated by, or special cases of, the other methods, so I dropped them from further study, focusing only on GA, SA, and TS. Of these SA is an old friend for statisticians, but TS is almost unknown in the statistics community and among many statisticians GA, for some reason, has a bad reputation. I found in the literature review that little was known about the optimal input settings for these three methods, so I conducted a large simulation experiment to investigate the quality of the solutions from each optimisation algorithm as a function of the method's inputs.

Using what worked best with $p = 14$ I then tackled the 83–variable case (Chapter 5). Instead of attempting an impossible task—complete full enumeration of all $2^p = 9.7 \cdot 10^{24}$ models—I created a proxy for truth by giving the best versions of each of GA, SA, and TS from the 14–variable runs each one week of CPU time (at 400 Unix MHz), collecting the 3,000 apparently best models found in this way, and performing a full enumeration on them. Finally, I conducted a limited simulation study of a number of the most promising input settings from the $p = 14$ case in the broader world of $p = 83$. Computing time constraints and the size of the solution space ensured that my work with $p = 83$ to date is only part of the story; I intend to continue this work for publication.

My main findings are as follows.

- As mentioned above, a method like the one used here, based on treating variable selection as a decision problem in a way that trades off data collection cost against predictive accuracy, can potentially save a great deal of money when the purpose of the model-building is the construction of a scale that will be used to predict outcomes for future individuals. This conclusion has wide implications (a) for variable selection in generalised linear models generally and (b) specifically in health policy, in the league-table quality assessment process described in Chapter 1.

- As is reasonable intuitively, the optimal choice of N is neither too small nor too large. Values between 2 and 10 work best in my problem; the best value varies with optimisation method but does not seem to depend strongly on p.

- The overall winner among the three major methods I examined was GA in both the 14– and 83–variable cases. However, GA is the method that changes its behaviour the most dramatically as you change the input settings: versions of GA had both the best and the worst performance with $p = 14$. More recent variations of GA, employing an elitist selection strategy (without mutation) and uniform or highly-uniform crossover, vastly outperformed the "vanilla" version of GA first proposed in the 1970s. Two other factors appear important to achieve good performance of GA in this problem: retention of 100% of the current members of the population at the beginning of each repetition of the algorithm, and a small to moderate population size. With $p = 83$ the highly uniform crossover strategy performed better than the uniform.

- TS comes in second to GA, and by an amount that seems to grow as p increases. TS has the advantage over GA of stability as a function of input settings: it is hard to make TS either very good or very bad by your choice of the inputs. To the extent that the inputs matter, it appears best to make the algorithm spend most of its time in the intensification search, followed by the preliminary search, and to spend the least time in diversification.

- It is interesting that the versions of GA that perform the best do so by (a) shutting off the mutation operation altogether and (b) keeping 100% of the current population as "start-up" individuals at each repetition of the algorithm, because both of these choices would seem to cut down on GA's ability to explore regions of the model space that differ sharply from those already examined. In the language of TS it is as though the optimal settings of GA choose an algorithm with a great deal of intensification and almost no diversification, which agrees at least partially with the previous conclusion.

- "Vanilla" SA may be an old friend, but it does not appear to be anywhere near the current best method for global optimisation in problems with binary inputs and multiple local maxima: it came a poor third to the other two methods both with $p = 14$ and 83, and the gap between it and the other two widened as p increased. For anybody who insists on using vanilla SA anyway, in this problem I found that (a) the logarithmic and reciprocal cooling schedules outperformed the geometric schedule which is so often favoured in the literature, and the straight schedule was by far the worst; (b) with $p = 83$ the logarithmic schedule was better than the reciprocal; and (c) $(T_0, T_f) = (1.0, 0.1)$ and $(0.5, 0.05)$ worked best as starting and finishing temperatures.

- Finally, it appears that GA gains its advantage over TS and SA by requiring less computational "overhead" in deciding where next to move: TS uses a lot of CPU time managing the tabu list, and the best cooling schedules for SA involve repeated expensive calls to the logarithm and exponential functions.

6.2 Suggestions for future work

I intend to continue this work in several directions as I move toward additional publications based on it. The following is a list of possible future work.

- Increase the size of the simulation experiment in the 83–variable case. It is possible that the best input settings with $p = 83$ may differ substantially from those in the 14–variable case. A more complete full-enumeration exercise than the one in Chapter 5 can be based on the observation from Table 5.1 that more than $\frac{2}{3}$ of the 83 variables never appear (and indeed that only 13 of them occur more than 10% of the time) in the 3,000 best models already found;

- Use an intelligent way to cut the neighbourhood size down from 83 to some much smaller number, in TS and SA, for the 83–variable case. In the present implementation of TS, when it is at a given model it has to evaluate 83 utilities to decide where to move next;

- Demonstrate the following point: if you want to *force* a given variable into the modelling this can easily be accommodated with this approach: instead of looking for the best subset from among the inputs $(x_1, \ldots, x_p)$, you force (say) x_1 into the model and look for the best subset from among the inputs $(x_2, \ldots, x_p)$ to add to x_1.

- Explore one or more hybrid strategies; here are two examples.

  - Given a budget of k hours, say, spend something like $0.7\,k$ running GA, and then spend the remaining time running something like TS to locally explore the best regions found by GA.
  - In SA, for instance: with $k = 3$ hours of CPU time, divide the first two hours (say) up into (say) four blocks of 30 minutes each; within each block use a different starting point; use a fairly large value of the final temperature like 0.1; get the 10 best models from each block; merge them and eliminate replicates to end up with (say) α models; and then use the final 1 hour to compute the $N = 500$ "real" utilities of the best α models (where α should be the appropriate number in order to do full-enumeration in 1 hour).

- See if parallelising the code for one or more of the optimisation methods results in an increase in performance.

- Try modifying an optimisation method such as TS in a way that is more directly informed by clinical judgement, e.g., swapping variables in and out of the model according to a grouping by body system (variables x_1 to x_{12} have to do with the lungs, x_{13} to x_{26} the heart, and so on).

- Explore different forms of the utility function altogether. Two examples are as follows.

  - The main purpose to which sickness-at-admission scales are to be put is in trying to identify good and bad hospitals by comparing observed mortality rates to expected rates given admission sickness (Chapter 2). In view of this I will try reformulating the problem so that utility is assessed at the hospital level rather than at the patient level, possibly leading to sickness scales that are even more relevant to health policy quality assessment.
  - Try a continuous utility function such as the log scoring rule mentioned in Chapter 2.

- How does the performance of this approach depend on the overall sample size n (in our case, 2,532)? It is intuitively reasonable that the quality of the decision about whether a hospital is "good" or "bad" would be lower with a much smaller data set. I could test this by setting up a simulation world in which I know the right answer and seeing how often the correct decisions are made as a function of n. More specifically, to do this right you would have to generate a lot of hospitals with varying quality of care and sickness at admission and see how many bad and good hospitals were correctly identified by input-output analysis based on the optimal sickness scale as a function of n.

- In SA there is a temperature interval in the middle in which the chain is moving around well (before this point it moves wildly from one bad model to another, after this point it gets stuck at whatever local mode it has found). To defeat this sort of behaviour I can look at the run from (say) iteration 1,000 to 5,000 in a 10,000–iteration run, find the best (say) 10 models during that interval, and either rapidly cool around them or revisit them with a much larger value of N.

- Do additional sensitivity analyses; for instance, with the C_{lm} in Table 2.1 in a way other than using the same multiplier across all four.

- Finally, I mentioned at the end of Chapter 1 that there were at least two distinct questions of interest here: (1) How well do some of the leading stochastic optimisation methods perform when they are guided by one or more ad hoc variable selection heuristics? (2) How well do such methods perform when they are not guided in this way? I have concentrated so far on question (2); in future work I will look at question (1). Here are two possibilities:

  - There is a big literature on variable selection in regression (ignoring data collection costs) which could provide ideas on how to focus the optimisation search (hints from stepwise regression methods, residual analysis, and so on). For example, suppose you have four variables you are already sure should be in the scale, and you are thinking about three new ones. By looking at the covariance matrix of all seven variables you can see if the new ones are likely to provide new information for predicting y above and beyond that already present in the old variables.

  - Using regression results on the "benefit" (only) of each variable to inform a method like TS when it is deciding on things like aspiration criteria.

  - It is not hard to construct measures of the desirability of a variable that trade off data collection costs and predictive accuracy in an ad hoc way. Table 6.1 presents one such measure in the 14–variable case. First, scale the marginal costs c_j by calculating $\frac{c_j}{\min c_j}$; small values are good. Then,

Table 6.1: *An ad hoc measure of the desirability of a variable in compromising between predictive accuracy and data collection costs, in the case $p = 14$.*

Variable	Cost c_j (US\$)	Correlation r_j	Good?	(1) $\frac{c_j}{\min c_j}$	(2) $\frac{\max r_j}{\lvert r_j \rvert}$	$d_j =$ (1) $\cdot$ (2)
APACHE II score	3.33	0.39		20	1.0	20.0
Age of patient	0.17	0.17	*	1	2.3	2.3
SBP score	0.17	0.29	**	1	1.3	1.3
CHF chest X-ray score	0.83	0.10		5	3.9	19.5
Blood urea nitrogen	0.50	0.32	**	3	1.2	3.7
APACHE II coma score	0.83	0.35	**	5	1.1	5.6
Serum albumin score	0.50	0.20	*	3	2.0	5.9
Shortness of breath	0.33	0.13	**	2	3.0	6.0
Respiratory distress	0.33	0.18	*	2	2.2	4.3
Septic complications	1.00	0.06		6	6.5	39.0
Prior resp. failure	0.67	0.08		4	4.9	19.5
Recently hospitalised	0.67	0.14		4	2.8	11.1
Ambulatory score	0.83	0.22		5	1.8	8.9
Initial temperature	0.17	−0.06	*	1	6.5	6.5

do something similar with the correlations with the ratio $\frac{\max r_j}{|r_j|}$; again the good variables are small on this. Now take the product d_j, as in the final column of the table. As it happens, this particular ad hoc desirability measure correlates well with whether or not a given variable appeared frequently in the 20 best models with $p = 14$; in fact, the predictors with the eight smallest values of d_j agree with the eight variables possessing one or more asterisks in Table 6.1. Of course, the only way we know this is to have gone through the exercise of maximising expected utility as in Chapters 2 and 4; other ad hoc measures might well look just as plausible, and how can we choose among them? Also, the last column in the table rank-orders the variables in desirability but says nothing about how many should be used to achieve the optimal tradeoff. Nevertheless, one possible application of the d_j, when normalised to probabilities in some way, would be to make a method like TS more intelligent in deciding which variable to bring next into the current model.

6.2.1 Improved simulated annealing

The performance of plain ("vanilla") simulated annealing in Chapters 4 and 5 was disappointing, so I decided to make another set of SA runs incorporating two improvements: the idea at the end of the previous section for teaching the stochastic optimisation methods about the desirability of the predictor variables, and the tabu search idea of random restarts. A detailed examination of a number of SA output files revealed that, even with temperature schedules that dropped the temperature slowly and with fairly high final temperatures, SA tended to get stuck in a local maximum of the criterion function fairly early in the run (e.g., as early as 1,500 model evaluations into a run with a planned 5,000 model evaluations). To try for better simulated annealing results I defined an *improved SA* (ISA) algorithm, as follows:

(1) ISA begins by choosing 20 models completely at random and evaluating their estimated expected utility (EEU) values using N^* replications (this is to initialise the league table of the 20 best models found so far).

(2) Next ISA starts the stochastic search at the null model (with no predictors), which becomes the current model, and computes its EEU value using the adaptive–N^* method (Section 4.5).

(3) ISA then begins proposing moves away from the current model using one-bit flips (at locations in the binary string governed by a *pointer* that scans from 1 to p and back again to 1) and the *variable desirability* criterion of Table 6.1. From the desirability values d_j in the last column of that table I created probabilities p_j^{in} and p_j^{out}—for flipping a 0 to a 1 and vice versa, respectively—using the transformation from desirability to probability given by

$$p_j^{\text{in}} = p_{\min} + (p_{\max} - p_{\min})\, e^{-c(d_j-1)}, \tag{6.1}$$

where $(p_{\min}, p_{\max}, c)$ are tuning constants to be specified by the user and p_j^{out} is simply taken to be $1 - p_j^{\text{in}}$. Here $p_{\min}$ and $p_{\max}$ govern how dogmatic the inclusion and exclusion processes should be, and c controls the rate at which desirability translates into probability of inclusion (recall that by construction small values of d_j represent greater desirability, and the smallest possible value is 1). Some experimentation led me to the choices $(p_{\min}, p_{\max}, c) = (0.1, 0.9, 0.1)$, which yielded the inclusion probabilities in Table 6.2 in the 14–variable case. A move away from the current model is then governed by two

Table 6.3: *Variable desirability values d_j and the corresponding p_j^{in} values in the 14-variable case, using the desirability-to-probability transformation given by equation (6.1).*

Variable (j)	d_j	p_j^{in}
APACHE II score	20.0	0.220
Age of patient	2.3	0.802
SBP score	1.3	0.876
CHF chest X-ray score	19.5	0.226
Blood urea nitrogen	3.7	0.711
APACHE II coma score	5.6	0.605
Serum albumin score	5.9	0.590
Shortness of breath	6.0	0.585
Respiratory distress	4.3	0.675
Septic complications	39.0	0.118
Prior resp. failure	19.5	0.226
Recently hospitalised	11.1	0.391
Ambulatory score	8.9	0.463
Initial temperature	6.5	0.562

processes in sequence: first a move is either proposed or not at random based on the p_j^{in} and p_j^{out} values, and then if a move is proposed it either takes place or not according to the usual SA acceptance probabilities. For example, with the p_j^{in} values in Table 6.2, if the current model is $(0, 0, \ldots, 0)$ ISA proposes a move to $(1, 0, \ldots, 0)$ with probability 0.220; suppose this proposal is turned down. ISA then proposes a move from $(0, 0, \ldots, 0)$ to $(0, 1, \ldots, 0)$ with probability 0.802; suppose this proposal is accepted. Then the move to this new model actually takes place or not with probability given by the usual SA acceptance regime based on the adaptive-N^* method for evaluating the EEU. And so on.

(4) Step (3) is repeated until the algorithm gets stuck in the same place for k consecutive iterations, where—again after some experimentation—I chose $k = 50$ as a good compromise between effective exploration of local optima and effective search of the whole space. If k successive steps without a move take place at any time during the run, ISA implements a random restart: the temperature is again set to T_0, a random initial model is generated, and cooling from this temperature begins all over again exactly as it did at the beginning of the entire algorithm. Throughout the run the league table of 20 best models is constantly updated.

(5) Steps (3) and (4) are then iterated until the desired amount of CPU time has been exhausted.

I performed a simulation experiment with ISA similar to the one with vanilla SA whose results were given previously in Section 4.6.2. As in the earlier experiment there were five user inputs to vary:

- r, the total number of loops from 1 to 14 in the location of the pointer used to propose a one-bit flip (this is chosen to achieve the specified target of CPU time);

- N^* (this varied from 2 to 20, as in vanilla SA);

- T_0 and T_f, the initial and final values of the temperature (these again varied across the five settings $(T_0, T_f) = (10.0, 1.0), (10.0, 0.1), (2.5, 0.1), (1.0, 0.1), (0.5, 0.05)$); and

- sc, the schedule used to decrease the temperature (as before 1 = straight, 2 = geometric, 3 = reciprocal, 4 = logarithmic).

I used the same 108 combinations of input settings (almost a full factorial) as with vanilla SA, each of which took approximately 1,200 seconds. The actual CPU time again varied by input settings, this time from a mean (across the 30 runs) of 1010 to 1695 seconds, so as before I calculated both raw summaries and results adjusted (via regression) for differences in CPU time. Tables 6.4–6.6 summarise the results, which are much better than those for vanilla SA: the best input settings achieve an adjusted mean value of p_{20}, the percentage of the actual 20 best models found in the run, of over 70%. For the very best input settings adjusted mean $p_{20} = 74.2\%$; the corresponding value for vanilla SA was 55.5%, and TS and GA only achieved 64.9% and 66.5%, respectively.

Some further conclusions emerging from Tables 6.4–6.6 are as follows.

- The logarithmic and reciprocal schedules performed best in the optimisation problem studied here using ISA: all 15 of the best input settings either had $sc = 4$ or 3 (12 of these were logarithmic), and all 16 of the worst input settings had $sc = 1$ or 2.

- The effect of the initial and final temperatures on performance was complicated and was linked (as was true with vanilla SA) to the cooling schedule: for example, 14 of the 16 worst input settings had $(T_0, T_f) = (10.0, 1.0)$ or

Table 6.4: *Results of the simulation study for improved SA with* $p = 14$ *(part 1). Values in parentheses are Monte Carlo standard errors; entries are sorted by adjusted means of* p_{20}.

					p_{20} (% of 20 Actual Best Models Found)		
				Mean CPU	Raw	Raw	Adjusted
N^*	T_0	T_f	sc	Time (sec)	Mean	SD	Mean
15	0.5	0.05	4	1349	0.748 (0.016)	0.114	0.742
20	0.5	0.05	4	1494	0.753 (0.017)	0.143	0.734
15	10	0.1	4	1360	0.715 (0.017)	0.089	0.708
20	10	0.1	4	1349	0.706 (0.017)	0.120	0.700
20	2.5	0.1	4	1495	0.708 (0.016)	0.110	0.689
10	0.5	0.05	4	1251	0.685 (0.018)	0.109	0.688
15	2.5	0.1	4	1407	0.698 (0.018)	0.090	0.687
10	10	0.1	4	1408	0.690 (0.018)	0.102	0.679
15	0.5	0.05	3	1127	0.655 (0.014)	0.088	0.669
10	1	0.1	4	1057	0.648 (0.017)	0.103	0.668
20	1	0.1	4	1278	0.660 (0.015)	0.141	0.660
10	0.5	0.05	3	1088	0.640 (0.017)	0.101	0.657
10	2.5	0.1	4	1453	0.671 (0.017)	0.090	0.656
15	1	0.1	3	1156	0.635 (0.021)	0.105	0.646
15	1	0.1	4	1032	0.620 (0.017)	0.120	0.642
10	0.5	0.05	1	1188	0.628 (0.013)	0.085	0.636
5	0.5	0.05	4	1126	0.608 (0.013)	0.078	0.622
10	0.5	0.05	2	1094	0.601 (0.025)	0.450	0.618
15	0.5	0.05	1	1129	0.598 (0.020)	0.104	0.612
20	0.5	0.05	3	1175	0.600 (0.020)	0.138	0.610
4	1	0.1	4	1056	0.590 (0.018)	0.101	0.610
5	10	0.1	4	1040	0.588 (0.014)	0.099	0.610
5	0.5	0.05	1	1205	0.601 (0.019)	0.085	0.608
5	1	0.1	4	1102	0.580 (0.018)	0.073	0.596
10	1	0.1	3	1010	0.571 (0.015)	0.088	0.595
20	0.5	0.05	2	1113	0.578 (0.015)	0.108	0.593
20	1	0.1	3	1138	0.578 (0.016)	0.110	0.591
5	0.5	0.05	3	1110	0.576 (0.019)	0.092	0.591
15	0.5	0.05	2	1091	0.566 (0.019)	0.104	0.583
20	0.5	0.05	1	1093	0.563 (0.015)	0.109	0.580
10	2.5	0.1	3	1371	0.588 (0.014)	0.087	0.580
5	2.5	0.1	4	1557	0.601 (0.014)	0.080	0.576
10	1	0.1	1	1179	0.565 (0.018)	0.094	0.574
5	0.5	0.05	2	1091	0.555 (0.014)	0.092	0.572
15	1	0.1	1	1318	0.563 (0.017)	0.094	0.565

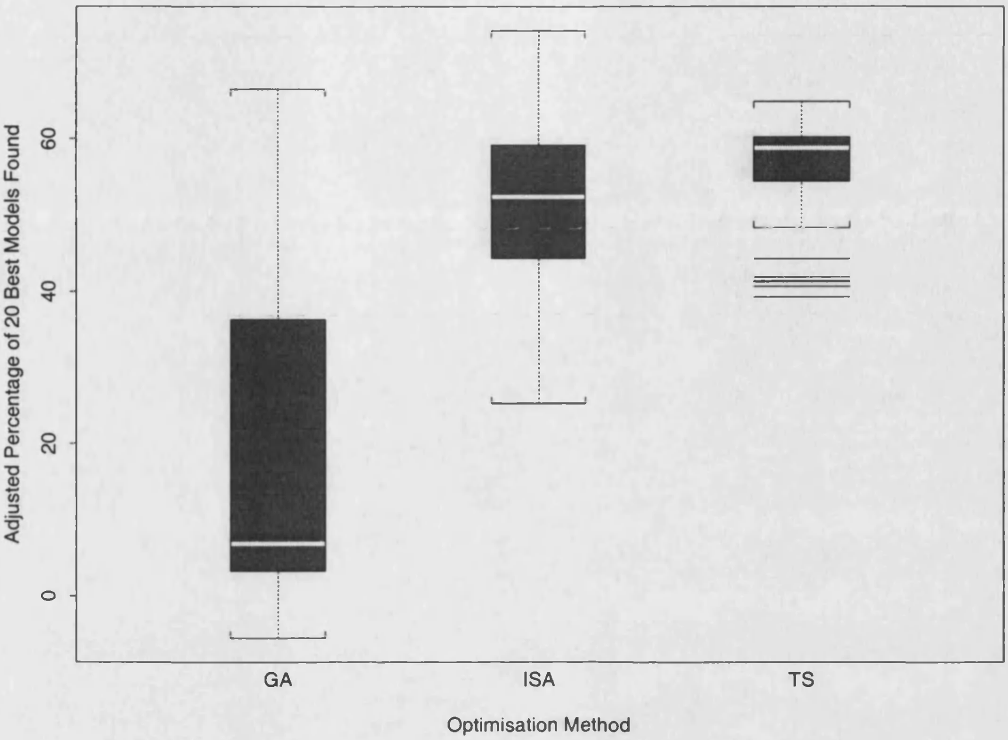
Table 6.5: *Results of the simulation study for improved SA with $p = 14$ (part2).*

					p_{20} (% of 20 Actual Best Models Found)		
				Mean CPU	Raw	Raw	Adjusted
N^*	T_0	T_f	sc	Time (sec)	Mean	SD	Mean
10	1	0.1	2	1159	0.550 (0.018)	0.086	0.561
15	10	0.1	3	1237	0.556 (0.018)	0.082	0.560
15	2.5	0.1	3	1318	0.560 (0.012)	0.103	0.557
5	1	0.1	1	1435	0.565 (0.014)	0.094	0.551
20	2.5	0.1	3	1296	0.550 (0.017)	0.108	0.549
3	1	0.1	4	1300	0.551 (0.016)	0.106	0.549
5	1	0.1	2	1219	0.540 (0.017)	0.077	0.546
2	1	0.1	4	1090	0.525 (0.016)	0.088	0.542
15	1	0.1	2	1210	0.535 (0.014)	0.108	0.541
2	0.5	0.05	4	1174	0.531 (0.022)	0.074	0.541
10	10	0.1	3	1407	0.546 (0.020)	0.946	0.535
20	1	0.1	1	1279	0.533 (0.012)	0.088	0.533
5	2.5	0.1	3	1373	0.541 (0.014)	0.081	0.533
4	1	0.1	3	1138	0.518 (0.014)	0.803	0.531
2	0.5	0.05	3	1154	0.518 (0.019)	0.072	0.530
5	1	0.1	3	1214	0.523 (0.018)	0.096	0.529
20	10	1	4	1464	0.543 (0.082)	0.075	0.527
2	0.5	0.05	1	1219	0.518 (0.016)	0.092	0.524
5	10	1	4	1412	0.536 (0.018)	0.080	0.524
5	10	0.1	3	1282	0.523 (0.020)	0.098	0.523
2	2.5	0.1	4	1409	0.531 (0.019)	0.101	0.520
2	10	0.1	4	1442	0.531 (0.016)	0.090	0.517
2	0.5	0.05	2	1232	0.510 (0.017)	0.101	0.515
4	1	0.1	1	1282	0.510 (0.146)	0.093	0.510
10	2.5	0.1	2	1367	0.518 (0.018)	0.066	0.510
4	1	0.1	2	1270	0.506 (0.013)	0.078	0.507
3	1	0.1	2	1259	0.500 (0.019)	0.103	0.502
15	10	1	4	1426	0.515 (0.020)	0.109	0.502
3	1	0.1	3	1269	0.500 (0.010)	0.101	0.501
3	1	0.1	1	1240	0.496 (0.013)	0.101	0.500
10	10	1	4	1695	0.535 (0.015)	0.084	0.498
10	2.5	0.1	1	1343	0.500 (0.022)	0.111	0.495
5	2.5	0.1	1	1418	0.506 (0.014)	0.099	0.494
2	1	0.1	3	1186	0.485 (0.172)	0.076	0.494
5	2.5	0.1	2	1403	0.496 (0.017)	0.077	0.485
2	1	0.1	1	1303	0.483 (0.015)	0.101	0.481
2	1	0.1	2	1298	0.481 (0.019)	0.077	0.480

Table 6.6: *Results of the simulation study for improved SA with* $p = 14$ *(part3).*

					p_{20} (% of 20 Actual Best Models Found)		
				Mean CPU	Raw	Raw	Adjusted
N^*	T_0	T_f	*sc*	Time (sec)	Mean	SD	Mean
20	10	0.1	3	1184	0.471 (0.018)	0.124	0.480
20	1	0.1	2	1174	0.468 (0.015)	0.107	0.478
10	10	1	3	1694	0.501 (0.014)	0.056	0.464
5	10	1	1	1483	0.480 (0.013)	0.086	0.462
10	10	1	2	1489	0.480 (0.025)	0.099	0.461
10	10	1	1	1603	0.483 (0.016)	0.098	0.454
2	2.5	0.1	3	1383	0.461 (0.018)	0.073	0.452
15	2.5	0.1	2	1309	0.450 (0.016)	0.111	0.448
2	2.5	0.1	1	1473	0.461 (0.013)	0.082	0.444
10	10	0.1	1	1201	0.435 (0.025)	0.097	0.442
5	10	0.1	1	1430	0.451 (0.021)	0.074	0.438
5	10	0.1	2	1449	0.451 (0.018)	0.093	0.436
20	10	1	3	1429	0.448 (0.013)	0.107	0.435
5	10	1	3	1352	0.441 (0.018)	0.076	0.435
5	10	1	2	1474	0.450 (0.019)	0.069	0.433
2	10	0.1	3	1562	0.458 (0.016)	0.085	0.433
15	2.5	0.1	1	1379	0.440 (0.013)	0.111	0.431
2	10	1	4	1490	0.438 (0.019)	0.075	0.419
2	10	1	3	1433	0.430 (0.015)	0.083	0.417
2	2.5	0.1	2	1477	0.430 (0.014)	0.077	0.413
15	10	1	3	1341	0.418 (0.013)	0.113	0.413
15	10	0.1	1	1255	0.408 (0.021)	0.116	0.410
15	10	1	1	1271	0.408 (0.016)	0.077	0.409
10	10	0.1	2	1149	0.395 (0.018)	0.097	0.407
20	2.5	0.1	1	1138	0.388 (0.018)	0.137	0.401
20	2.5	0.1	2	1124	0.380 (0.016)	0.124	0.394
2	10	0.1	2	1487	0.411 (0.020)	0.081	0.393
2	10	0.1	1	1400	0.403 (0.016)	0.076	0.392
15	10	1	2	1454	0.403 (0.016)	0.101	0.388
15	10	0.1	2	1073	0.361 (0.014)	0.089	0.380
2	10	1	1	1377	0.381 (0.018)	0.094	0.373
2	10	1	2	1350	0.378 (0.026)	0.082	0.372
20	10	1	1	1096	0.320 (0.020)	0.104	0.337
20	10	1	2	1174	0.305 (0.019)	0.097	0.315
20	10	0.1	1	1084	0.273 (0.014)	0.096	0.291
20	10	0.1	2	1073	0.233 (0.013)	0.096	0.252

Figure 6-1: *Parallel boxplots comparing GA, ISA, and TS in the 14-variable case.*



(10.0, 0.1) (when straight or geometric schedules were used), but two of the top four input settings also had $(T_0, T_f) = (10.0, 1.0)$ (when the schedule was logarithmic).

- In general low values of N for the logarithmic and reciprocal schedules, and high values of N for the straight and geometric schedules, performed badly with ISA. The worst combination was a high value for the initial temperature together with large N for the straight and geometric schedules. It is noteworthy in comparing ISA to vanilla SA that large values of N work so well with ISA when the best cooling schedule is chosen (all 16 of the best input settings have $N \geq 10$). This appears to be due not so much to the random-restart feature in ISA as to the use of desirability to search for good variables to include in the model.

- With the modifications of SA involving random restarts and the inclusion and exclusion of variables based on desirability, the new ISA outperforms both of the versions of TS and GA studied in Chapter 4 in the $p = 14$ case. Figure 6.1 gives parallel boxplots of the results from GA, ISA, and TS across all input configurations examined; the maximum across the three methods was

Table 6.7: *Input settings and results for the 3-hour runs of ISA in the simulation experiment with* $p = 83$ *(SDs in parenthesis).*

	Run									
Input	1	2	3	4	5	6	7	8	9	10
N^*	15	20	15	20	20	10	15	10	15	10
T_0	0.5	0.5	10.0	10.0	2.5	0.5	2.5	10.0	0.5	1.0
T_f	0.05	0.05	0.1	0.1	0.1	0.05	0.1	0.1	0.05	0.1
Cooling Schedule	4	4	4	4	4	4	4	4	3	4

	Run									
Result	1	2	3	4	5	6	7	8	9	10
Mean CPU (10K sec)	0.88	0.96	1.12	1.11	0.77	1.08	0.86	0.88	0.98	1.12
# $(N = 1)$	4642	5326	8108	7995	4718	5169	5554	8426	6727	7053
# $(N > 1)$	212	533	109	97	107	273	124	187	109	120
Mean* Dimension	16.3 (3.5)	14.4 (4.3)	19.9 (2.4)	19.0 (1.4)	15.7 (2.4)	18.1 (2.1)	17.1 (2.6)	19.4 (1.1)	19.6 (1.2)	15.9 (3.8)
Min* Dimension	0.8	1.0	1.0	1.0	1.4	1.0	1.0	1.0	1.0	4.6
Max* Dimension	32.6	29.0	35.0	32.2	29.6	29.8	32.2	31.8	33.4	26.6
Mean* Apparent Utility	−10.9	−10.2	−11.4	−10.9	−10.2	−11.0	−10.6	−11.3	−11.1	−10.2
Mean** Real Utility	−8.5	−7.8	−8.6	−8.2	−8.1	−8.3	−8.2	−8.8	−8.6	−8.3
Mean** Dimension	7.3 (1.4)	6.6 (0.7)	6.4 (1.4)	5.0 (1.2)	4.9 (1.1)	6.3 (2.8)	5.1 (1.2)	7.5 (2.9)	6.6 (1.5)	7.3 (3.4)
# 6** Best	3	5	3	3	3	3	3	4	3	3
Max Efficiency (1K)	0	14.7	0	0	2.9	8.3	2.9	0	0	0

attained by ISA, although TS still has the largest median and the smallest variability around the median (which gives some idea of typical performance of the algorithms without a lot of fine-tuning). This comparison is not quite fair to TS and GA, because the versions studied earlier did not have the benefit of the desirability idea. In any case ISA is perhaps best viewed as a hybrid method, incorporating ideas both from SA and TS, and it would seem that if

> a great deal of hybridising experimentation is allowed all three methods would be likely to perform about the same (in fact, if you combine ideas from several methods it is no longer even meaningful to talk about a contest between the three approaches).

Table 6.7 summarises the performance of the improved SA method in the $p = 83$ case (naming conventions for the inputs are as above). As with GA, TS, and ordinary SA, I took the ten best input configurations from the $p = 14$ case and made five runs with different random number seeds, in each case allowing a budget of three hours of CPU time. In the runs reported on here N^* varied from 10 to 20, I used two different cooling schedules (logarithmic in most of the cases, and in one case reciprocal) and the initial and final temperatures ranged from 0.5 to 10.0 and 0.05 to 0.1, respectively. The mean CPU times actually observed varied from 7,700 to 11,200 seconds. The total number of models visited by ISA ranged from 4,854 to 8,613 (much better than vanilla SA), and in most of the cases the great majority of these models were examined with $N = 1$.

It is evident from the mean real utility and mean dimension rows of this table that the two new ideas in ISA have led to a dramatic improvement over vanilla SA; in fact, ISA performs about as well as GA (the previously best method) with $p = 83$. In future work I intend to quantify how much of ISA's improvement is due to the random restart idea and how much to the introduction of the desirability criterion.

Bibliography

Aarts EHL, Korst JHM (1989). *Simulated Annealing and Boltzmann Machines.* Chichester: Wiley.

Ackley D (1987). *A Connectionist Machine for Genetic Hill-climbing.* Dordrecht: Kluwer.

Alander JT (1992). On optimal population size of genetic algorithms. *Proceedings CompEuro 92*, 65–70. IEEE Computer Society Press.

Baker JE (1985). Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (Davis L, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Beaty S (1991). *Instruction Scheduling using Genetic Algorithms.* Ph.D. Dissertation, Colorado State University.

Becker RA, Chambers JM, Wilks AR (1993). *The New S Language.* Pacific Grove, CA: Wadsworth & Brooks/Cole.

Bergeret F, Besse P (1997). Simulated annealing, weighted simulated annealing and genetic algorithm at work. *Computational Statistics*, **12**, 447–465.

Bernardo JM, Smith AFM (1994). *Bayesian Theory.* New York: Wiley.

Beyer D, Ogier R (1991). Tabu learning: a neural network search method for solving nonconvex optimization problems. *Proceedings of the International Joint Conference on Neural Networks.* Singapore: IEEE and INNS.

Bland JA, Dawson GP (1991). Tabu search and design optimization. *Computer-Aided Design*, **23**, 195–202.

Booker LB (1987). Improving search in genetic algorithms. In *Genetic Algorithms and Simulated Annealing* (Davis L, ed.). San Mateo, CA: Morgan Kaufmann.

Brandimarte P, Conterno R, Laface P (1987). FMS production scheduling by simulated annealing. *Proceedings of the 3rd International Conference on Simulation in Manufacturing*, 235–245.

Caruna RA, Schaffer JD (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. *Proceedings of the fifth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Cohoon JP, Hegde SU, Martin WN, Richards D (1987). Punctuated equilibria: a parallel genetic algorithm. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Connoly DT (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research*, **46**, 93–100.

Chakrapani J, Skorin–Kapov J (1993). Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, **41**, 327–341.

Chams M, Hertz A, de Werra D (1987). Some experiments with simulated annealing for colouring graphs. *European Journal of Operational Research*, **32**, 260–266.

Chatfield C, Collins AJ (1980). *Introduction to Multivariate Analysis.* London: Chapman & Hall.

Chipman JS, Winker P (1995). Optimal industrial classification by threshold accepting. *Control and Cybernetics*, **24**, 477–494.

Cvijovic D, Klinowski J (1995). Tabu search: An approach to the multiple minima problem. *Science*, **267**, 664–666.

Davis L (1991). *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

Daley J, Jencks S, Draper D, Lenhart G, Thomas N, Walker J (1988). Predicting hospital-associated mortality for Medicare patients with stroke, pneumonia, acute myocardial infarction, and congestive heart failure. *Journal of the American Medical Association*, **260**, 3617–3624.

De Jong KA (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Ph.D. Dissertation, University of Michigan.

de Werra D, Hertz A (1989). Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum*, **11**, 131-141.

Donabedian A (1981). Advantages and limitations of explicit criteria for assessing the quality of health care. *Milbank Memorial Fund Quarterly—Health and Society*, **59**, 99–106.

Downsland KA (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, **68**, 389–399.

Draper D (1995). Inference and hierarchical modeling in the social sciences (with discussion). *Journal of Educational and Behavioral Statistics*, **20**, 115–147, 233–239.

Draper D (1996). Hierarchical models and variable selection. Technical Report, Department of Mathematical Sciences, University of Bath, UK.

Draper D, Fouskakis D (2000). A case study of stochastic optimization in health policy: problem formulation and preliminary results. *Journal of Global Optimization*, **18**, 399–416.

Draper D, Kahn K, Reinisch E, Sherwood M, Carney M, Kosecoff J, Keeler E, Rogers W, Savitt H, Allen H, Wells K, Reboussin D, Brook R (1990). Studying the effects of the DRG-based Prospective Payment System on Quality of Care: Design, sampling, and fieldwork. *Journal of the American Medical Association*, **264**, 1956–1961.

Dueck G, Scheuer T (1990). Threshold acceptance: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, **90**, 161–175.

Eshelman L (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms* (Rawlins G, ed.). San Mateo, CA: Morgan Kaufmann.

Fairley A (1991). *Comparison of methods of choosing the crossover point in the genetic crossover operation.* Department of Computer Science, University of Liverpool.

Fiechter CN (1994). A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, **51**, 243–267.

Fouskakis D, Draper D (1999). Review of *Tabu Search*, by F Glover and M Laguna, Amsterdam: Kluwer (1997). *The Statistician*, **48**, 616–619.

Franconi L, Jennison C (1997). Comparison of a genetic algorithm and simulated annealing in an application to statistical image reconstruction. *Statistics and Computing*, **7**, 193–207.

Geman S, Geman D (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI–6**, 721–741.

Gilks WR, Richardson S, Spiegelhalter DJ (1996). *Markov Chain Monte Carlo in Practice.* London: Chapman & Hall.

Glauber RJ (1963). Time–dependent statistics of the Ising model. *Journal of Mathematical Physics*, **4**, 294–307.

Glover F (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, **8**, 156–166.

Glover F (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, **13**, 533–549.

Glover F (1989). Tabu search–Part I. *ORSA Journal on Computing*, **1**, 190–206.

Glover F (1990). Tabu search–Part II. *ORSA Journal on Computing*, **2**, 4–32.

Glover F (1990). Tabu search: A tutorial. *Interfaces*, **20**, 74–94.

Glover F, Glover R, Klingman D (1986). The threshold assignment algorithm. *Mathematical Programming Study*, **26**, 12–37.

Glover F, McMillan C (1986). The general employee scheduling problem: an integration of management science and artificial intelligence. *Computer and Operational Research*, **15**, 563–593.

Glover F, Taillard E, de Werra D (1993). A user's guide to tabu search. *Annals of Operations Research*, **41**, 3–28.

Goldberg DE (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison–Wesley.

Goldberg DE, Deb K (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms* (Rawlins G, ed.). San Mateo, CA: Morgan Kaufmann.

Goldberg DE, Deb K, Kargupta H, Harik G (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Illigal Report No. 93004*, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana.

Goldberg DE, Deb K, Korb B (1990). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, **4**, 415–444.

Goldberg DE, Korb B, Deb K (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, **3**, 493–530.

Goldberg DE, Lingle R (1985). Alleles, loci and the travelling salesman problem. In *Genetic Algorithms and Their Applications: Proceedings of the Second*

International Conference on Genetic Algorithms (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Goldberg DE, Richardson J (1987). Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Goldstein H, Spiegelhalter DJ (1996). League tables and their limitations: Statistical issues in comparisons of institutional performance (with discussion). *Journal of the Royal Statistical Society, Series A*, **159**, 385–444.

Grefenstette JJ (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, **16(1)**, 122–128.

Hadorn D, Draper D, Rogers W, Keeler E, Brook R (1992). Cross-validation performance of patient mortality prediction models. *Statistics in Medicine*, **11**, 475–489.

Hansen P (1986). The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods in Combinatorial Optimization.* Capri, Italy.

Hansen P, Jaumard B (1990). Algorithms for the maximum satisfiability problem. *Computing*, **44**, 279–303.

Hertz A, de Werra D (1987). Using tabu search techniques for graph coloring. *Computing*, **29**, 345–351.

Holland JH (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Hosmer DW, Lemeshow S (1989). *Applied Logistic Regression.* New York: Wiley.

Inshibuchi H, Misaki S, Tanaka H (1995). Modified simulated annealing algorithms for the flow shop sequencing problem. *European Journal of Operational Research*, **81**, 388–398.

Jaumard B, Hansen P, Poggi di Aragao M (1991). Column generation methods for probabilistic logic. *ORSA Journal on Computing*, **3**, 135–148.

Jencks S, Daley J, Draper D, Thomas N, Lenhart G, Walker J (1988). Interpreting hospital mortality data: The role of clinical risk adjustment. *Journal of the American Medical Association*, **260**, 3611–3616.

Jennison C, Sheehan N (1995). Theoretical and empirical properties of the genetic algorithm as a numerical optimizer. *Journal of Computational and Graphical Statistics*, **4**, 296–318.

Jog P, Suh JY, Gucht DV (1989). The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In *Proceedings of the Third International Conference on Genetic Algorithms* (Schaffer JD, ed.). San Mateo, CA: Morgan Kaufmann.

Johnson DS, Aragon CR, McGeoch LA, Schevon C (1989). Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operational Research*, **37**, 865–892.

Kahn K, Brook R, Draper D, Keeler E, Rubenstein L, Rogers W, Kosecoff J (1988). Interpreting hospital mortality data: How can we proceed? *Journal of the American Medical Association*, **260**, 3625–3628.

Kahn K, Rogers W, Rubenstein L, Sherwood M, Reinisch E, Keeler E, Draper D, Kosecoff J, Brook R (1990). Measuring quality of care with explicit process criteria before and after implementation of the DRG-based Prospective Payment System. *Journal of the American Medical Association*, **264**, 1969–1973.

Kahn K, Rubenstein L, Draper D, Kosecoff J, Rogers W, Keeler E, Brook R (1990). The effects of the DRG-based Prospective Payment System on quality of care for hospitalized Medicare patients: An introduction to the series. *Journal of the American Medical Association*, **264**, 1953–1955 (with editorial comment, 1995–1997).

Kapsalis A, Smith GD, Rayward-Smith VJ (1993). Solving the graphical Steiner tree problem using genetic algorithms. *Journal of the Operational Research Society*, **44**, 44.

Keeler E, Kahn K, Draper D, Sherwood M, Rubenstein L, Reinisch E, Kosecoff J, Brook R (1990). Changes in sickness at admission following the introduction of the Prospective Payment System. *Journal of the American Medical Association*, **264**, 1962–1968.

Kelley A, Pohl I (1995). *A Book on C*, third edition. Redwood City, CA: Benjamin/Cummings.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

Kitano H (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, **4**, 461–476.

Knaus WA, Draper EA, Wagner DP, Zimmerman JE (1985). APACHE II: A severity of disease classification system for severely ill patients. *Critical Care Medicine*, **13**, 818–829.

Knuth DE (1968). *The Art of Computer Programming. Volume 1: Fundamental Algorithms.* Reading, MA: Addison-Wesley.

Kvasnička V, Pospíchal J (1995). Messy simulated annealing. *Journal of Chemometrics*, **9**, 309–322.

Laguna M, Barnes JW, Glover F (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, **2**, 63–74.

Laguna M, Glover F (1993). Bandwidth packing: A tabu search approach. *Management Science*, **39**, 492–500.

Laguna M, Gonzalez-Velarde JL (1991). A search heuristic for just–in–time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, **2**, 253–260.

Lundy M, Mees A (1986). Convergence of an annealing algorithm. *Mathematical Programming*, **34**, 111–124.

Malek M, Guruswamy M, Pandya M, Owens H (1989). Serial and parallel simulated annealing and tabu search algorithms for the travelling salesman problem. *Annals of Operations Research*, **21**, 59–84.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1091.

Meyer TP, Packard NH (1992). Local forecasting of high-dimensional chaotic dynamics. In *Nonlinear Modeling and Forecasting* (Casdagli M, Eubank S, ed.). Reading, MA: Addison-Wesley.

Michalewicz Z, Janikow CZ (1991). Genetic Algorithms for numerical optimization. *Statistics and Computing*, **1**, 75–91.

Montana DJ, Davies LD (1989). Training feedforward networks using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Mühlenbein H, Gorges-Schleuter M, Krämer O (1988). Evolution algorithms in combinatorial optimization. *Parallel Computing*, **7**, 65–85.

Ogbu FA, Smith DK (1990). The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem. *Computers and Operational Research*, **17**, 243–253.

Oliveira S, Stroud G (1989). A parallel version of tabu search and the path assignment problem. *Heuristics for Combinatorial Optimization*, **4**, 1–24.

Pettey CB, Leuze MR, Grefenstette JJ (1987). A parallel genetic algorithm. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Rawlins GJE (1991). *Foundations of Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Reeves CR (1992). A genetic algorithm approach to stochastic flowshop sequencing. *Proceedings of the IEEE Colloquium on Genetic Algorithms for Control and Systems Engineering.* Digest No. 1992/106, London: IEEE.

Reeves CR (1995). *Modern Heuristic Techniques for Combinatorial Problems.* London: McGraw-Hill.

Schaffer JD, Caruana RA, Eshelman LJ, Das R (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms* (Schaffer JD, ed.). San Mateo, CA: Morgan Kaufmann.

Schultz–Kremer S (1992). Genetic algorithms for protein tertiary structure prediction. In *Parallel Problem Solving from Nature 2* (Männer R, Manderick B, ed.). North-Holland.

Sechen C, Braun D, Sangiovanni-Vincetelli A (1988). Thunderbird: A complete standard cell layout package. *IEEE Journal of Solid-State Circuits*, **SC-23**, 410–420.

Semet F, Taillard E (1993). Solving real–life vehicle routing problems efficiently using tabu search. *Annals of Operations Research*, **41**, 469–488.

Sirag DJ, Weisser PT (1987). Towards a unified thermodynamic genetic operator. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Skorin–Kapov J (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, **2**, 33–45.

South MC, Wetherill GB, Tham MT (1993). Hitchhiker's guide to genetic algorithm. *Journal of Applied Statistics*, **20**, 153–175.

Stander J, Silverman BW (1994). Temperature schedules for simulated annealing. *Statistics and Computing*, 4, 21–32.

Stata Reference Manual, Release 5. College Station, TX: Stata Press.

Suh JY, Van Gucht D (1987). Incorporating heuristic information into genetic search. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (Grefenstette JJ, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Syswerda G (1989). Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (Schaffer JD, ed.). San Mateo, CA: Morgan Kaufmann.

Taillard E (1989). *Parallel tabu search for the job shop scheduling problem.* Research Report ORWP 89/11. DMA Lausanne, Switzerland: EPFL.

Taillard E (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, **47**, 65–74.

Taillard E (1991). Robust tabu search for the quadratic assignment problem. *Parallel Computing*, **17**, 443–455.

Talbi EG, Bessière P (1991). A parallel genetic algorithm applied to the mapping problem. *SIAM News*, July 1991, 12–27.

Tipler PA (1969). *Foundations of Modern Physics.* New York: Worth.

Tovey CA (1988). Simulated simulated annealing. *AJMMS*, **8**, 389–407.

Tukey JW (1977). *Exploratory Data Analysis.* Reading, MA: Addison-Wesley.

Ulder NLJ, Aarts EHL, Bandelt HJ, Van Laarhoven PJM, Pesch E (1991). Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Solving from Nature* (Schwefel HP, Manner R, ed.). Berlin: Springer.

Vakharia AJ, Chang YL (1990). A simulated annealing approach to scheduling a manufacturing cell. *Naval Research Logistics*, **37**, 559–577.

Van Laarhoven PJM, Aarts EHL (1988). *Simulated Annealing and Boltzmann machines.* Chichester: Wiley.

Volker N, Henrik P (1995). A modification of threshold accepting and its application to the quadratic assignment problem. *OR Spektrum*, **17**, 205–210.

Weisberg S (1985). *Applied Linear Regression*, second edition. New York: Wiley.

Whitley D (1989). The GENITOR algorithm and selection presure: why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms* (Schaffer JD, ed.). San Mateo, CA: Morgan Kaufmann.

Whitley D (1992). *Foundations of Genetic Algorithms 2.* San Mateo, CA: Morgan Kaufmann.

Widmer M (1991). Job shop scheduling with tooling constraints: a tabu search approach. *Journal of Operation Research Society*, **42**, 75–82.

Winker P, Fank KT (1997). Application of threshold accepting to the evaluation of the discrepancy of a set of points. *SIAM Journal on Numerical Analysis*, **34**, 2028–2042.