UNIVERSITY OF SOUTHAMPTON

Parameter Optimisation for Search Heuristics via a Barrier Tree Markov Model

by

William Benfold

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

February 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by William Benfold

The quality of solution provided by a search heuristic on a particular problem is by no means an absolute value. Most heuristics are controlled by a set of parameters, upon which the performance is heavily dependent. We construct Markov models for search heuristics on specific problem instances, and model the relationship between the quality of search and the choice of parameters.

For any problem instance of nontrivial size, the state space for this model is large enough so as to make computation infeasible. Our solution is to use a reduced-state model of the search space, by amalgamating many similar points. We use a model based on a level-accessible barrier tree, grouping regions of the search space according to the reachability of minima.

Optimal annealing and mutation schedules are produced, minimising either "where-you-are" or "best-so-far" cost, over binary perceptron, spin-glass and Max-SAT problems. The predictions of the model are found to be consistently over-optimistic; we discuss reasons for this and suggest some possible refinements to the model.

A population-based variant of simulated annealing is briefly examined, where annealing temperature is adjusted according to performance. We later optimise the average first-passage time for several special-case heuristics, comparing the minimal times across a range of problems.

Contents

List of Figures

Nomenclature

$\mathbb{R}$	The set of reals
$\mathbb{R}^+$	The set of strictly positive reals
$\mathbb{Z}$	The set of integers
$\mathbb{Z}^+$	The set of strictly positive integers (equivalently $\mathbb{N}$)
$\mathbf{0}$	A (column) vector of zeros (number of elements to be inferred from context)
$\mathbf{1}$	A (column) vector of ones
$\lfloor x \rfloor$	The "floor" function, given by $\max\{y \in \mathbb{Z} : y \leq x\}$
$\lceil x \rceil$	The "ceiling" function, given by $\min\{y \in \mathbb{Z} : y \geq x\}$
$[a, b]$	The closed interval $\{x : x \in \mathbb{R}, a \leq x \leq b\}$
$[pred]$	Evaluates to 1 if *pred* is true, 0 otherwise
$\mathbb{P}(X)$	The probability of event X occurring
$\langle X \rangle$	The arithmetic mean of the random variable X
$P(\mathcal{S})$	The power set of $\mathcal{S}$
$M^\top$	The transpose of the matrix M
$A \otimes B$	The element-wise product of the matrices A and B

Acknowledgements

The Max-SAT, binary perceptron and spin-glass problem sets, and associated Barrier Tree data (state sizes, costs and transition probabilities) used in this thesis were kindly contributed by Jonathan Hallam. §4.3 provides a brief overview of the process, which is described in full in Hallam's thesis (Hallam, 2006).

Chapter 1

Introduction

The efficiency of search heuristics when applied to real-world problems is of considerable importance, both to the practitioners, who wish to maximise the return on invested resources, and to theorists interested in studying the difficulty of the problems themselves.

A practitioner is faced with the task of applying search heuristics in such a way as to produce solutions of highest quality, whilst minimising resource consumption (usually execution time). Since these two objectives will almost always conflict, one variable is usually predetermined, while the other is optimised; one is usually interested in either the best solution produced within some time limit, or the time required to reach a solution of at least a certain quality.

Regardless of the exact specification of the goal, the behaviours of all but the simplest of search heuristics are determined in part by a set of parameters[1]. For instance, Simulated Annealing is controlled by a set of real-valued parameters known as the "annealing schedule". A genetic algorithm may have such parameters as "mutation rate", "crossover probability", "population size", and some more structured information, such as the choice of selection method and crossover operator, whether a generational or steady-state GA is used, and so on.

From the point of view of one analysing the complexity of hard problems, optimised heuristics can be used to measure the difficulty of the problems themselves. The performance of optimised heuristics on various problems can give a quantitative measure of the relative difficulties of the problems.

Finding the best choice of parameters for a given heuristic is usually a task considerably more difficult than solving the problem to optimality; the chief benefits are indirect,

[1]Of course, unparameterised search heuristics do exist, with first-improvement descent and exhaustive (or even random) search being common examples. In fact, the size of the parameter set of any algorithm may be increased or decreased arbitrarily via generalisation, specialisation, or various other mechanisms. Our description of the parameter set for any algorithm is merely intended to describe the "natural parameterisation", and is of course not derived from any inherent property of that algorithm.

through application to related problems. It is not unreasonable to expect that when a series of search problems are constructed in a similar manner (e.g. a set of 20-variable Max-3-SAT problems, a set of 15-variable spin-glass problems, or a set of 30-node TSP problems), some aspects of the structure of the problem landscape (e.g. the size, depth, number or clustering of minima, plateaus and barriers) are common to, or at least correlated amongst, that series of similar problems. If a set of problems have some similarity of structure, one could expect that heuristic behaviour, and thus the best choice of heuristic parameters, might be similar across those problems. Optimal parameters for one problem in such a set, while unlikely to be optimal for the other problems, may provide improved performance over parameter settings chosen without this analysis. It is possible that the benefits might outweigh the additional resource expenditure required for the optimisation, particularly in scenarios where "training problems" are made available in advance of the real problem set.

It is also possible that, given a problem with a search space too large to analyse, a smaller problem from the same class may be generated (or the original problem projected in some way onto a smaller space[2]), with the intention that applying the search to the reduced problem may provide some insight into the original problem. Even if the smaller problem bears no semblance to the original, the results of applying a given heuristic on the two may be related, and the relationship possibly predicted through analysis of the way the difficulty of the problem scales with size.

1.1 Overview

Fig. 1.1 outlines the process. Given a search problem, a model of the corresponding cost landscape is produced. This model represents the landscape as a number of states, each with a single cost, and with transition probabilities reflecting the probability of moving between them under a random walk. This is the output of a *state-amalgamation* process, produced either through knowledge of the structure of the space gleaned from the problem definition, or by constructing a specific set of equivalence classes known as a *barrier tree* (§4.3) for the landscape (the steps involved are general enough that they may be applied to arbitrary cost functions over finite, discrete spaces). The cost landscape model is intended to be transparently substituted in many situations where one would normally require a complete model of the search space.

The behaviour of a search heuristic can be defined independently of the landscape on which it is searching, provided obviously that no representation-specific operations are performed (e.g. computing directions between points). We also restrict the choice of heuristics to those for which the state space is identical to the search space, i.e. those where the algorithm can be said to be "at" a particular point in the search space after

[2]In fact, we do exactly this when we use state amalgamation to reduce the size of a problem (§4.3).

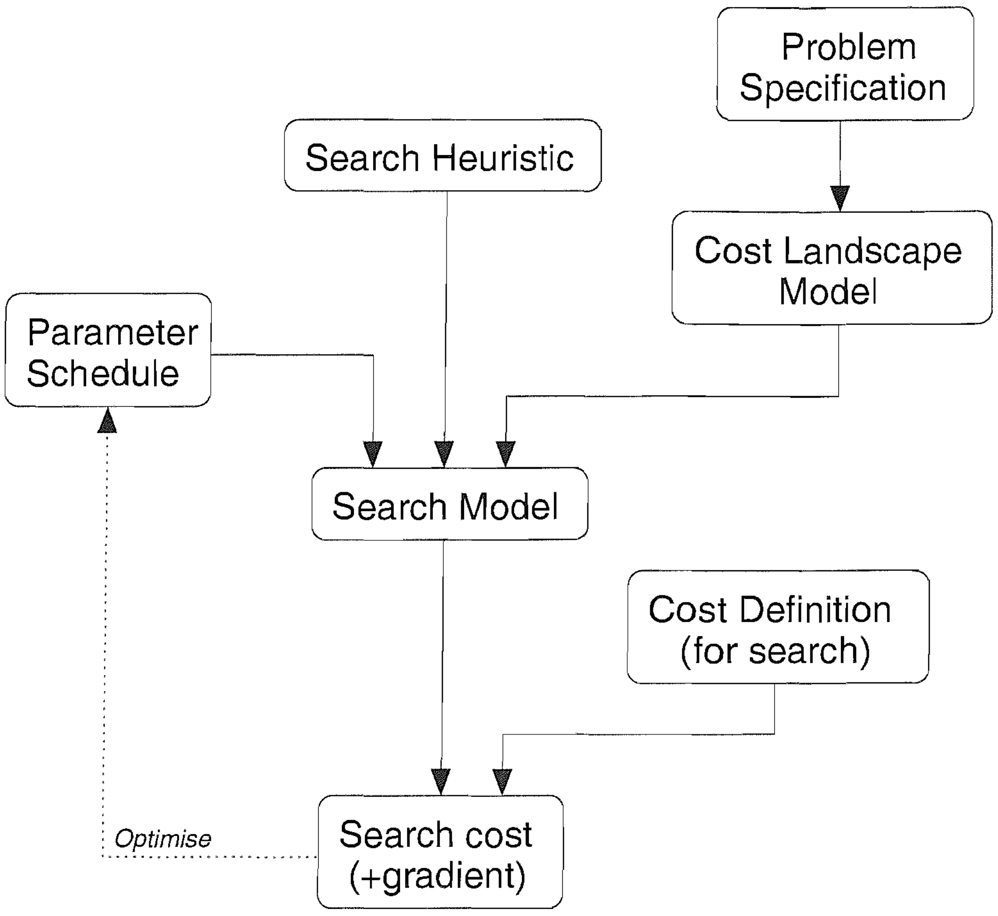

FIGURE 1.1: Overview of search parameter optimisation

each iteration. Notably, this excludes most GAs; for a population of size P, with each individual represented by L bits, the size of the state vector would be $O(2^{LP})$.

Combined with a cost landscape model, this description of a heuristic's behaviour can be used to model the path taken through the search space as a Markov process, and to compute the expected cost. The behaviour of the search is dependent upon the search parameters (e.g. an annealing schedule), as is, by extension, the average cost. Our work centres around the optimisation of search cost through the parameters of the heuristic.

1.2 Existing Work

1.2.1 Models of Search Spaces

The idea of representing a search space as a hierarchy of states is certainly not new, but the method of decomposition according to accessibility was recently introduced by Flamm et al. (2002). Further exploration of barrier tree construction was performed in Hallam and Prügel-Bennett (2003, 2005b); Hallam (2006), with barrier trees constructed for several NP-hard problems. Hallam used Markov models constructed from the barrier tree data to predict the final distribution and average search time for first-improvement descent, and in Hallam and Prügel-Bennett (2005a) attempted to produce a barrier-tree based model for crossover, using the average Hamming distance between states to calculate the probability distribution for offspring.

(Our results for the Max-SAT, binary perceptron and spin glass problems are based on barrier tree data contributed by Jonathan Hallam, consisting of transition probabilities, state sizes and costs for a series of problem instances from each class.)

Blaudeck and Hoffmann (2003) modelled SA on a continuous problem (minimum-energy arrangements of carbon atoms) by partitioning the landscape into local minima (further grouped according to energy level), and measuring the escape time and distribution of destination states for each minimum at a variety of temperatures.

Courtois (1985) provides a useful discussion on modelling nearly-decomposable complex systems. Our model could be seen as a simple two-level hierarchical system; having divided the landscape into equivalence classes, we model the intra- and inter-class probability flow separately. The inter-class model is the Markov model we construct for the search; the intra-class model is that used by Hallam to predict the distribution of exit points and average length of a random walk within each class.

Courtois produced bounds for the steady-state probabilities of Markov "subchains", reasoning that the behaviour complex hierarchical systems is often dominated by that of a small number of subsystems. A variety of approaches for modelling the interaction between these critical subsystems and the rest of the system are presented in Courtois and Semal (1986); the more information available on these interactions, the better the approximation.

Prior to state-aggregation, to apply Courtois' methods to our system would be difficult. There are many (e.g. 2^{20}) states to consider, and no convenient way to determine which subset should be considered "critical". Indeed, there is no reason to assume that there exists a subset of states with substantially greater influence on a heuristic, nor that such a set would be small enough to model directly.

After the search space has been reduced to a barrier tree model (or even to the intermediate collection of level-connected sets: §2.1.2), the system may be more amenable to Courtois' form of analysis. The barrier tree model groups search points according to their reachability (via descent) from each other; it is not unreasonable to assume that the resulting groups interact particularly weakly. The problem is that the connectivity of the state-aggregated system is much less regular than that of the original. Although the number of states is substantially reduced, the transitions between states cannot (in the general case) easily be modelled *en masse*, and must be considered individually.

State aggregation methods need not be confined to the search space model; Spears and De Jong (1997) applied several aggregation schemes to a model of a GA. The states of the GA model (i.e. the possible populations) were grouped according to fitness, homogeneity (the degree of correlation between members of a population) and probability mass, with low-probability states being discarded. The results showed showed a 90% reduction in the number of model states, with as little as a 1% loss in accuracy.

Our approach differs in that the aggregation is applied directly to the search space, producing a heuristic-independent model of the problem. This has the advantage of clearer mapping between model states and landscape features. The downside is that the problem of rapid growth in the state space of models of population-based heuristics is left unaddressed, leaving our models restricted mostly to "single point" heuristics such as SA and variable mutation search. Our search space reduction is also much more extreme, with over a million states reduced to between twenty and two hundred. This has undoubtedly cost us much accuracy (§5.3), but has allowed us, along with our choice to look only at heuristics with minimal state, to model significantly larger problems, based on 20 and even 40 bit strings, compared with the 2 and 3 bit strings used by Spears and De Jong (1997).

1.2.2 Models of Search Heuristics

Hajek (1988) studied annealing schedules which were optimal in the sense of maximising probability of reaching the global minimum in the infinite time limit. Hajek proved a condition (necessary and sufficient) for an annealing schedule to reach the optimum with certainty in this limit.

Hoffmann and Salamon (1990) modelled SA on a four-state problem, producing schedules with optimal final energy (equivalent to WYA cost: §3.2.1). This was followed by a scaling analysis (Christoph and Hoffmann, 1993) of optimal annealing schedules on two fifteen-state problems, where rescaled schedules of increasing length were found to converge.

Strenski and Kirkpatrick (1991) found a similar relationship, with annealing schedules optimised for a five-state graph bisection problem following a "decreasing envelope which is relatively independent of the length". Strenski also compared the performance of several commonly used schedule types (linear, geometric, inverse logarithmic) on a 200-state 1D random energy problem. It was found that the inverse logarithmic schedule, whilst having the theoretically attractive property of maximising the probability of traversing the highest barrier, was outperformed on this problem by the other two schedule types.

Boese and Kahng (1994) introduced a model for "best-so-far" cost (BSF), which we use in §3.2.2. Their results also demonstrated (on a six-node TSP problem and on the same graph bisection problem used by Strenski.) that optimising "where-you-are" (WYA) cost is not sufficient to provide a good quality BSF schedule. We confirm their results for a 16-bit hurdle problem in §5.2.

Recognising that "convergent" schedules were often impractical for application to real problems, Cohn and Fielding (1999) compared a variety of non-convergent schedule types on instances of the TSP problem (containing between 48 and 442 cities). The best choice of parameters for each schedule was found experimentally (the schedules were each controlled by only two or three parameters); the performance of each schedule was then evaluated using various cost measures (BSF, FPT, probability of reaching (or of achieving a cost within 1% of) the global minimum).

White and Mayne (2000) looked at a kinetic system with a small (two, three, four) number of energy levels, with transitions between levels requiring a barrier to be overcome. In each case, the goal was to maximise the probability of being at the ground state on the final iteration. For the two-level system, an optimal annealing schedule for the problem was found analytically. On all problems, a series of optimal schedules (for different model parameters and time limits) were produced using a GA; the two-level GA schedules were found to match the analytical result.

White's choice of representation for the GA deserves further comment. Schedules were assumed to be monotonically decreasing, and were permitted only a finite set of temperature values. Rather than having each site of the GA chromosome represent the temperature at a particular iteration (i.e. optimising across finitely many real valued parameters, as we do in §5), the chromosome had one (boolean) site for each possible temperature, indicating whether that temperature should appear in the schedule or should be skipped. This representation has the advantage of allowing a mutation to "insert" a series of steps at arbitrary points in the schedule. Unfortunately this schedule form is not suitable in every case; we shall see in §5 that, as several others have noted, the optimal schedules for some problems are non-monotonic.

Optimal schedules may of course be produced for arbitrary problems, without a model of the search space, if one is willing to use sampling to determine schedule performance. The penalty is an increase in execution time, as even a single sample of the search space may be more computationally expensive than evaluating a well-chosen model. In fact, not only is each iteration of the schedule optimise more time-consuming, but the number of iterations is likely to increase too, due to the noisy evaluation provided by the sampling. Nonetheless, the prospect of producing optimal schedules for arbitrary problems is tempting. Bölte and Thonemann (1996) attempted to use Genetic Programming to optimise schedules for quadratic assignment problems in up to 50 variables. Several types of schedules resulted, some showing oscillating or periodically spiking temperature.

The variable mutation search we later model (§3.1.3) could be described as a $(1+1)$ Evolutionary Algorithm, i.e. an EA with a population of one, producing one mutated offspring per generation[3], with the best of the two surviving to the next generation. The (1 + 1) EA has also been analysed in some depth, with expectation (or upper bounds thereof) calculated by Droste et al. (2002) for the average FPT of this algorithm on a variety of problems.

1.2.3 Summary

Modelling (and optimising parameters for) heuristics on real problems is a difficult task. In most of the existing work, this difficulty is overcome by reducing one aspect of the problem to near triviality. In some cases the heuristic parameters are constrained by assuming a particular form of schedule; sometimes the search performance is not modelled, but sampled experimentally and optimised directly. In some cases the problems examined are extremely small, or are structured so as to simplify analysis. Indeed, much time has been devoted to problem landscapes chosen chiefly for their analytical properties.

Our work is an attempt to apply some of the existing methodology to models of small (but nontrivial) instances of "real" (e.g. NP-hard) problems. In §5, we produce "freely optimised" annealing and mutation schedules; artificial constraints on the schedules values are kept to a practical minimum[4]. We produce closed-form expressions for the search cost, mostly through necessity, as the high dimensionality of the parameter space makes optimisation difficult without an accurate gradient. The simplifications we choose to make all this practical are the state-aggregation we apply to the search space (§4.3), and our choice to concentrate on "single-point" heuristics, i.e. those with a state space identical to the search space.

1.3 Roadmap

In §2, we shall properly explain such terms as "cost landscape" and "search problem", and formally specify the simulated annealing and descent with variable mutation algorithms.

In §3, we construct Markov models for these heuristics. The states of the models represent points (or collections thereof) in the search space, while each transition matrix represents one iteration of the algorithm, and will vary depending on the search parameters used. The model itself is therefore best described not a Markov chain, but as a series of Markovian transformations. We also introduce two different measures of cost for a search (§3.2), both specified in terms of the output of the Markov model. Thus we

[3]The attentive reader will later note that our definitions produce a slightly different distribution of offspring than the usual formulation.

[4]In most cases, the only constraint is that temperature changes may only occur every fourth iteration.

have the cost of a search as a function of the search parameters. Both simulated annealing and descent with variable mutation have sufficiently structured parameter sets that each transition matrix depends on exactly one parameter, which influences no other iterations. As a result, the derivative of cost with respect to the entire parameter set may be computed without too much extra effort, thus enabling the use of a gradient-based method to solve the continuous optimisation problem of finding the best parameters.

We take advantage of the simplicity of the parameter space once again in §3.3, where we introduce coarse-grained schedules. A parameter schedule may be approximated by grouping the parameters into fixed-size blocks (the approximation is later tested (§5.1.5), and appears to be valid, with the optimised coarse-grained schedules resembling "sampled" versions of a non-coarse-grained optimised schedule).

A collection of "test problems" is defined in §4, comprising one "toy" problem and three sets of small instances of NP-hard problems. The former is a problem with a large (in fact, maximal) number of local minima, known as the "hurdle problem" (defined in Prügel-Bennett (2004b)). We exploit the high degree of symmetry in the search space for the hurdle problem to produce an abstracted representation with only a handful of states. Despite this reduction, the model is still "exact" insofar as the predicted cost can be proven to be identical to that of the original model. Without this reduction in size, the matrices in the Markov model would be so large as to make manipulation infeasible.

The other problems are small instances of traditional "hard problems", which do not possess such obvious structural symmetry, and thus cannot be reduced so trivially. Again, the transition matrices in the Markov model become unmanageably large for all but the tiniest of problems; the state vector for the Markov model still contains the same number of elements as the search space for the problem. Thus each transformation of the state vector requires $O(2^{2L})$ operations to compute. We combat this by constructing a smaller model of the search space from a "barrier tree" model (Hallam (2006), summarised in §4.3); this reduced model has significantly fewer states. The barrier tree data (state sizes, costs, transition probabilities) used in our experiments was generated by Jonathan Hallam.

The complexity (and computational cost) of our minimisation therefore depends on the size of the barrier tree of the problem. Although the size of the barrier tree will grow with that of the problem, the reduction in the number of states is quite significant: a 20-variable Max-SAT problem is typically reduced from a state space of over a million points to a 30-40 node barrier tree. Although this is still only a very small Max-SAT problem by the standards of those who specialise in solving Max-SAT, it is considerably larger than the typical problems for which optimal annealing schedules are usually constructed.

Optimised annealing and mutation schedules for two cost functions and a variety of problems are compared in §5. The performance of these schedules on the actual problems shows qualitative agreement with the predictions of the model, but with a consistent over-optimism which we attribute to the loss of information in substituting a barrier tree model for the problem. In §5.7 we present results for an annealing schedule optimised over a set of Max-SAT problems.

§6 explores a population-based variant of Simulated Annealing, where a group of "annealers" explore a search space independently, but each have a temperature influenced by performance relative to the rest of the population. Unfortunately, experiments show no improvement to the search other than that which would be gained by having an equal number of annealers with identical fixed annealing schedules. Reasons for this and possible modifications and extensions to the model are discussed in §6.4.

Finally, in §7, we introduce a series of "toy algorithms" - heuristics which operate with *a priori* knowledge of the search space. For these, we compare the average First-Passage Times (§7.2) with those of single-parameter versions of simulated annealing and mutation search (§7.5). By looking at the performance of these special-case algorithms, we hope to gain some insight as to the features of a problem which present a challenge to heuristics.

Parts of this thesis have been submitted to Theoretical Computer Science (Benfold et al., 2005b), and have been presented at the IEEE CEC 2005 (Benfold et al., 2005a).

Chapter 2

Background

This chapter is intended to provide formal definition of (and limited discussion on) items of terminology which will be used later. Most of the definitions (with the probable exception of level-connectedness (§2.1.2)) are sufficiently commonplace that most readers may skip to the next chapter.

2.1 Cost Landscapes

We start by considering the generalised idea of a "problem". In order for a problem to be well-defined, there must be some criterion for establishing whether a given object is the "solution", and this criterion must itself be well-defined. We describe this formally by saying that any problem can be represented by a function of the form $f : \mathcal{S} \to \{\mathit{true}, \mathit{false}\}$. $\mathcal{S}$ is known as the search space; an element $s \in \mathcal{S}$ is said to be a solution of f iff $f(s) = \mathit{true}$.

We shall be concerned only with decidable problems; that is, those for which f may be computed algorithmically. For instance, the question "Does there exist a proof of theorem X?" has a trivially small search space of $\{\mathit{yes}, \mathit{no}\}$, but the test for a solution is clearly not decidable in the general case. By common convention, a solution to the problem without proof that f is satisfied would not be considered to be a complete solution. Roughly speaking, a problem is said to be a *search problem* if the task of proving that f is satisfied is trivial compared to that of finding the correct value s. For example, NP-hard problems fall into this category, as instances are (generally) not solvable in polynomial time[1], yet the correctness of a proposed solution to any instance *can* be checked in polynomial time.

[1]Assuming P $\neq$ NP

2.1.1 Minimisation Problems

A minimisation problem is a search problem with a solution criterion of the form $f(s) = (\forall t \in \mathcal{S}, c(s) \leq c(t))$, for some function $c : \mathcal{S} \rightarrow \mathcal{C}$, known as the *cost function*; $\mathcal{C}$ may be any ordered set. Although any problem could be trivially reformulated as a minimisation problem, through a substitution of $\{0, 1\}$ for $\{\mathit{true}, \mathit{false}\}$, we shall reserve the term for those problems with solution criteria defined "naturally" in terms of a minimisation.

We define a *neighbourhood function* on the space to be a function $n : \mathcal{S} \rightarrow P(\mathcal{S})$, where $P(\mathcal{S})$ denotes the power set of $\mathcal{S}$. The *neighbourhood* of a point s is given by $n(s)$, which is required to be "reciprocal" insofar as $x \in n(y) \iff y \in n(x)$; thus n may be used to construct an undirected graph of the connectivity of $\mathcal{S}$.

It is usually taken forgranted that a class of problems presented as minimisation problems have cost and neighbourhood functions with a sufficiently structured relationship so that there exists an algorithm for finding the minimum which requires (on average) fewer iterations than an exhaustive search. However, the "No Free Lunch Theorem" (Wolpert and Macready, 1995) tells us that this seemingly-modest condition cannot always hold; averaging over all possible cost functions for a search space, even the cleverest neighbourhood definition (or even an algorithm for choosing a neighbourhood definition) cannot possibly yield any net benefit.

One could say that in the general case, no neighbourhood is useful, all heuristics are futile, and all problems are difficult. The performance of a search depends on cost function, neighbourhood and heuristic, and it is not usually possible to say anything useful about the optimality/difficulty of one parameter unless the other two have been fixed (or at least specialised). In this thesis, we concentrate on the third, the heuristic; we look at problem instances individually, and assume that neighbourhoods are described as part of the problem specification. The practical reality is that most "real-world" search problems are defined over spaces on which there already exist neighbourhood definitions which are both familiar and lend themselves easily to machine representation.

Let a *maximisation problem* be defined in a similar way, and let an *optimisation problem* be the union of the two classes. Since any maximisation problem can, without loss of structure, be rewritten as a minimisation problem, we will assume that all optimisation problems are in fact minimisation problems.

2.1.2 Landscape Terminology

A set of points is said to be connected iff, for any two points in that set, there exists a path between those two points. Formally, $\mathcal{X} \subset \mathcal{S}$ is connected iff

$$\begin{array}{ll} (\forall x, y \in \mathcal{X})\,(\exists L \in \mathbb{Z}^+ \cup \{0\})\,(\exists p : \mathbb{Z} \to \mathcal{X}) & \\ \text{s.t.} \quad p(0) = x \quad \wedge \quad p(L) = y & . \\ \text{and} \quad \forall i, j \in \mathbb{Z}, \quad 0 \le i < j \le L \implies p(i) \in n(p(j)) & \end{array}$$

Two points are said to be *level-connected* if there exists a path between them on which all points (including the start and endpoints) have identical cost; the definition above may be modified for level-connectedness by adding the extra condition $(\forall i)\, c(p(i)) = c(p(0))$.

It is not difficult to see that level-connectedness is an equivalence relation on $\mathcal{S}$. Reflexivity is given by the trivial path, symmetry is inherent in the definition (as n is reciprocal, x and y may be exchanged without consequence), and transitivity results from path concatenation (two uniform-cost paths sharing a node must possess the same cost).

The equivalence classes for level-connectivity are known as *level-connected sets*; the cost of each such set is simply that of any of its elements (since they will all be of equal cost). We can extend the idea of a neighbourhood to level-connected sets: two level-connected sets are neighbours if at least one point in one set is a neighbour of at least one point in the other. So

$$\mathcal{X} \in N(\mathcal{Y}) \iff (\exists x \in \mathcal{X})\,(\exists y \in \mathcal{Y})\,(x \in n(y))\,.$$

Observe that no level-connected set can have a neighbour[2] of equal cost; if this were so, there would exist a (trivially short) path between the sets with constant cost, and thus a pair of points, one from each set, would be equivalent. By transitivity, all points in the union of the two classes would be equivalent, and thus the two classes could not be distinct.

A level-connected set for which all the neighbours are of higher cost is known as a *minimum*. A minimum with the lowest cost is known as a *global* minimum, with all others being *local* minima. There will always exist a global minimum, although there is no requirement that it be unique.

We use the term "plateau region" to describe a level-connected set of non-trivial size which is *not* a minimum. Traversal of plateau regions can be awkward, as the points in the set are indistinguishable; the behaviour on a plateau can usually be modelled as a

[2] We adopt the convention that a neighbour of a point x is any point y such that $x \in n(y)$ and $x \neq y$. A point may or may not be included in its own neighbourhood, but will not be said to be a neighbour of itself.

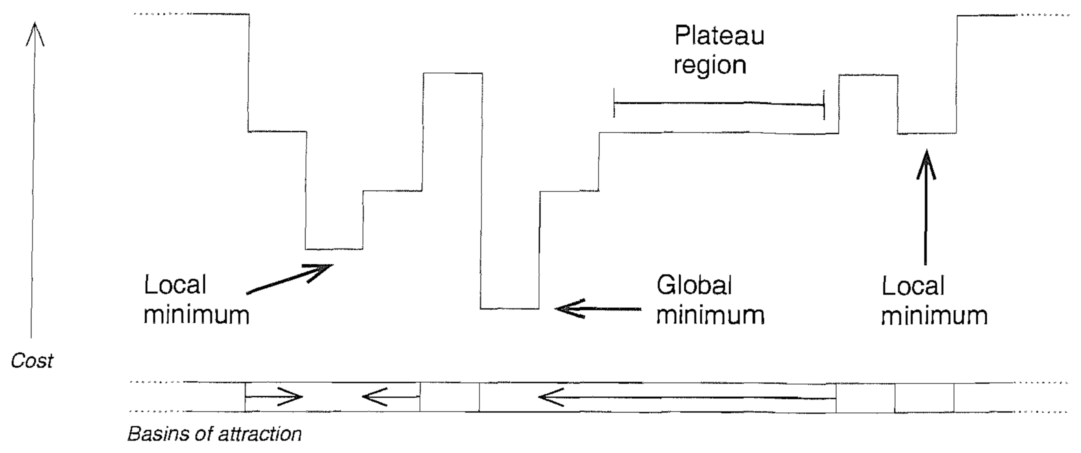

FIGURE 2.1: A one-dimensional cost landscape with several minima. Locations of minima and the associated basins are marked, as is a plateau region.

random walk (§A.5). This effect is greatest when the shape of the set is such that the internal connectivity is maximised, especially when interior points are created (those for which all neighbours are members of the set).

We shall occasionally speak of the "basin" around a minimum; by this we mean the set of points from which no other minima may be reached along a path of non-increasing cost. Equivalently, this is the region from which a descent (§2.2.1) is guaranteed to (eventually) reach that minimum. Note that no two basins intersect; a search point may be contained within at most one basin. A search point which is not within any basin is called a *saddle point.*

Basins, plateaus and minima are illustrated in fig. 2.1 using a one-dimensional cost landscape.

2.2 Search Heuristics

A *search heuristic* is an algorithm for solving optimisation problems which has no *a priori* information about the structure of the problem, or of any specific properties it may have. The following basic actions are typically available to a search heuristic:

Cost evaluation: For any $s \in \mathcal{S}$, the cost $c(s)$ may be computed

Neighbour selection: A point may be randomly[3] chosen from $n(s)$

[3]The distribution is usually assumed to be uniform, although we do not assert that this must be the case - all we require is that the heuristic is not able to influence the distribution in any way. In some situations, a non-uniform neighbour selection scheme may aid the search; we shall simply assume that this is implemented transparently (if at all).

The heuristic is supplied with one starting point and is expected to move towards the global optimum. The exact goal often varies; in some scenarios (§7.2), the optimum must be reached at all costs, in others (§3.2.2) the heuristic should attempt to arrive at a reasonably low-cost point as soon as possible. A key point is that the behaviour of the heuristic is completely abstracted from the representation of the problem. The two actions described above are the *only* valid operations which may be performed upon an element of the search space.

In some problem domains, extra operations may be added which make explicit use of properties of the representation. For instance, genetic algorithms use a "crossover" operator, allowing two points to be combined in some way to produce a single "child" point. This operation is usually implemented by randomly choosing an element from a subspace described by the two "parent" points; the relationship between this subspace and the parents is invariably tied to the neighbourhood connectivity of the search space, and thus encompasses problem-specific assumptions. Where available, a good recombination operator can result in a more efficient search, but it cannot easily be generalised to arbitrary problems.

Typically, performing one or both of the actions above will implicitly consume some resource, usually execution time. The performance of a heuristic is therefore based on a comparison of solution quality versus "time" taken. We shall assume that the time taken for cost evaluation dominates that of neighbour selection. The former shall be said to take one unit of time, the latter takes no time at all. Since most of the heuristics we consider perform exactly one cost evaluation per iteration, the terms "time unit" and "iteration" shall often be used interchangeably.

2.2.1 Descent

First-improvement descent (which we shall refer to simply as "descent") is amongst the simplest of practical minimisation algorithms. Each iteration of the algorithm (alg. 1) involves choosing a random neighbour of the "current" point, and moving to it if it has cost no worse than the current point. Note that requiring a new point to have a strictly lower cost in order to be adopted would cause the algorithm to become stuck at "plateau" points, for which all neighbours are of equal cost. Allowing same-cost transitions permits escape from these points.

Note that the number of cost evaluations is equal to the number of iterations, as $c(x)$ is always known from the previous iteration (or given from the initial conditions).

A variation on this algorithm is "best-improvement" descent, where *all* the neighbours of the search point are evaluated, with the lowest-cost neighbour being selected as the new search point. The obvious disadvantage is the increase in the number of cost evaluations required, but there is also the more subtle effect of decreasing variation over different

Algorithm 1: First-improvement descent

```
Assumption: x is a random point in S, c(x) is known
while termination criterion not satisfied do
    tmp ← random-neighbour(x);
    if $c(tmp) \leq c(x)$ then
        x ← tmp;
    end
end
```

runs. Two best-improvement descents started from the same point will travel along identical paths, except for where several neighbours share the lowest cost, in which case one of these will be chosen at random. Where a neighbour point is a local minimum, and is the lowest-cost neighbour, it will be chosen every time. Thus the chance of eventually reaching the global minimum might be reduced in favour of maximising the immediate improvement in cost.

Various modifications are also possible, with the results sometimes overlapping with other types of algorithm. History information may be used to prevent repetition (reducing the time spent in plateau regions); this falls into the broad category of Tabu Search (Glover, 1986). Because it is possible to detect when a descent has reached a local minimum (and is therefore unable to offer any further improvement), it is not unusual to either restart the algorithm ("iterated descent"), or to give the search a "kick" to a point some distance away from the current one. Doing so allows descent to make use of all the time available.

2.2.2 Descent with Variable Mutation

Descent is prone to becoming "stuck" at local minima; if a point is reached where all neighbours have higher cost, the search will progress no further. One modification which prevents this "stagnation" is the introduction of transitions to states outside of the local neighbourhood. Instead of selecting a random neighbour of the current point, the current point is "mutated" a number of times. Each mutation is simply a transition to a random neighbour. The multiply-mutated point is then adopted if it has a cost no worse than that of the original point. The number of mutations is Poisson-distributed, with the expectation controlled by a "mutation rate" parameter; a mutation rate equal to one therefore does not give the same behaviour as first-improvement descent.

2.2.3 Simulated Annealing

Simulated Annealing (Kirkpatrick et al., 1983; Cerny, 1985) is a heuristic inspired by the physical process of annealing, in which some material (usually a metal) is heated and then gradually cooled. Increasing the thermal energy allows bonds to be broken

Algorithm 2: Descent with variable mutation

```
Assumption: x is a random point in $S$, $c(x)$ is known
$t \leftarrow 1$;
while termination criterion not satisfied do
    $tmp \leftarrow x$;
    $m \leftarrow$ a Poisson deviate with expectation $u(t)$;
    for $i \leftarrow 1$ to $m$ do
        $tmp \leftarrow$ random-neighbour($tmp$);
    end
    if $c(tmp) \leq c(x)$ then
        $x \leftarrow tmp$;
    end
    $t \leftarrow t + 1$;
end
```

and reformed easily enough that any regular structure disappears. A slowly decreasing temperature allows time for the least-energy arrangements to form; this usually produces localised crystalline growth. By contrast, a sudden decrease in temperature (known as a "quench") removes the energy before it can be used to free the defects, resulting in crystalline regions with dislocations. The combination of structural properties desired is used to determine the function of temperature against time, known as the "annealing schedule".

The algorithm (alg. 3) can be thought of as a version of descent which sometimes permits uphill steps, depending on "temperature", with higher temperatures relaxing the usual prohibition on uphill moves. Where a higher-cost point is found, it may be accepted with a probability dependent upon the cost difference from the current point, and a parameter β. The usual[4] formulation of a simulated annealing algorithm uses a Boltzmann-distributed acceptance probability, inspired by the model of Metropolis et al. (1953) for the motion of atoms in a fluid under the effects of a heat bath. The Boltzmann parameter β is known as the "inverse temperature" by analogy to physical annealing, and is usually varied over time; the set of values for β is known as the *annealing schedule*. If the temperature is reduced to zero ($\beta \to \infty$), uphill transitions are never accepted, and the algorithm reduces to descent. When temperature is very large ($\beta \to 0$), all neighbours are accepted indiscriminately, producing a random walk.

2.2.4 Termination Criteria

In most cases, one chooses to terminate a search either after a specified number of iterations have been performed, or when a solution of sufficiently low cost has been found.

[4]An alternative acceptance strategy known as "Threshold Annealing" was introduced by Dueck and Scheuer (1990), whereby an uphill transition is permitted iff the energy (cost) difference is below some threshold.

Algorithm 3: Simulated Annealing

```
Assumption: $x$ is a random point in $S$, $c(x)$ is known
$t \leftarrow 1$;
while termination criterion not satisfied do
    $tmp \leftarrow$ random-neighbour($x$);
    if $c(tmp) \leq c(x)$ then
        $x \leftarrow tmp$;
    else
        /* Selected point was uphill, so randomise decision */
        Let $p$ be chosen randomly from $[0, 1)$;
        if $p < \exp(\beta(t)(c(x) - c(tmp)))$ then
            $x \leftarrow tmp$;
        end
    end
    $t \leftarrow t + 1$;
end
```

We shall be mostly interested in fixed-length parameter schedules (the first case), although in §7.2, we examine the average first passage time to the global minimum. In practical situations, the termination criterion need not be so simple, and could incorporate any available information about the search. For instance, the solution quality from previous iterations may be used to estimate the likelihood (and magnitude) of future improvement.

2.3 Linear Algebra 101

Let M be a square matrix of size $n \times n$. Then a vector $\mathbf{v}$ is said to be a (right-handed[5]) *eigenvector* of M if $M\mathbf{v} = \lambda\mathbf{v}$ for some scalar λ, known as the *eigenvalue* associated with $\mathbf{v}$. Rearranging this definition, we can produce

$$(M - \lambda I)\,\mathbf{v} = \mathbf{0}$$

where I is the $n \times n$ identity matrix. From this it can be seen that $(M - \lambda I)$ is not invertible[6], and thus $|M - \lambda I| = 0$. Evaluating this determinant for a general matrix M will produce a polynomial of degree n in λ (known as the characteristic equation of M), and thus there are at most n distinct eigenvalues for M. The number of times an eigenvalue λ appears as a root of the characteristic equation is known as the "algebraic multiplicity" of λ. A repeated eigenvalue will often have multiple linearly independent eigenvectors associated with it. The space of all possible eigenvectors of λ is known as the *eigenspace* of λ; the dimension of this space is referred to as the "geometric multiplicity" of λ, and is always at least one, and no greater than the algebraic multiplicity. Where it

[5]Unless specified otherwise, we take the term "eigenvector" to mean "right-handed eigenvector".

[6]Consider $\mathbf{v} = (M - \lambda I)^{-1}\,\mathbf{0}$. The zero vector remains unchanged when multiplied by any matrix, so no choice of inverse satisfies the equation (except for the degenerate case where $\mathbf{v} = \mathbf{0}$).

is less, the eigenvalue (and the matrix itself) are said to be *defective* (§B); in this case, an eigenvector basis for the space on which the matrix acts cannot be found.

If the eigenvectors of a matrix have no repeated eigenvalues, then they are linearly independent. This may be demonstrated by induction on the size of the eigenvector set (the base case is trivial, a single vector is clearly a linearly independent set of vectors). Let $\mathbf{v}_1 \ldots \mathbf{v}_k$ be a set of k linearly independent eigenvectors of M, with corresponding eigenvalues $\lambda_1 \ldots \lambda_k$. Now suppose $\mathbf{v}_{k+1}$ is an eigenvector with eigenvalue λ_{k+1}, but is also a linear combination of $\mathbf{v}_1 \ldots \mathbf{v}_k$:

$$\begin{aligned}
\mathbf{0} &= (M - \lambda_{k+1} I)\, \mathbf{v}_{k+1} \\
&= (M - \lambda_{k+1} I) \sum_{i=1}^{k} \alpha_i \mathbf{v}_i \\
&= \sum_{i=1}^{k} \alpha_i\, (M - \lambda_{k+1} I)\, \mathbf{v}_i \\
&= \sum_{i=1}^{k} \alpha_i\, (\lambda_i - \lambda_{k+1})\, \mathbf{v}_i.
\end{aligned}$$

If λ_{k+1} is a unique eigenvalue, then $(\lambda_i - \lambda_{k+1})$ will always be nonzero, and thus the vectors $\mathbf{v}_1 \ldots \mathbf{n}_k$ cannot be independent, violating the inductive assumption. Thus $\mathbf{v}_{k+1}$ must be independent of the other eigenvectors.

The "left-handed" eigenvectors are defined similarly, with $\mathbf{u}$ being a LH eigenvector if $\mathbf{u}^\top M = \lambda \mathbf{u}^\top$. The LH and RH eigenvectors share the same eigenvalues, as the polynomial defining them is identical. Observe that left- and right-handed eigenvectors are orthogonal unless they share eigenvalues:

$$\begin{array}{rrcl}
 & (\mathbf{u}_i^\top M)\mathbf{v}_j & = & \mathbf{u}_i^\top (M\mathbf{v}_j) \\
\Longrightarrow & (\lambda_i \mathbf{u}_i^\top)\mathbf{v}_j & = & \mathbf{u}_i^\top (\lambda_j \mathbf{v}_j) \\
\Longrightarrow & (\lambda_i - \lambda_j)\mathbf{u}_i^\top \mathbf{v}_j & = & 0
\end{array}$$

2.3.1 Eigenvalue Decomposition

Given an $n \times n$ matrix M and n linearly-independent eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, we produce a matrix V where each column is an eigenvector, i.e.

$$V = \begin{bmatrix} \mathbf{v}_1 & \ldots & \mathbf{v}_n \end{bmatrix}.$$

Let Λ be a diagonal matrix of the λ_i (i.e. $\Lambda_{ij} = [i = j]\lambda_i$); we observe that

$$MV = \begin{bmatrix} \lambda_1 \mathbf{v}_1 & \dots & \lambda_n \mathbf{v}_n \end{bmatrix} = V\Lambda$$

and M can be rewritten as $V\Lambda V^{-1}$. Thus, once a full set of eigenvectors has been computed, powers of M may be computed efficiently through powers of the eigenvalues:

$$\begin{aligned} M^a &= \left(V\Lambda V^{-1}\right)^a \\ &= \left(V\Lambda V^{-1}\right)\left(V\Lambda V^{-1}\right)\dots\left(V\Lambda V^{-1}\right) \\ &= V\Lambda\left(V^{-1}V\right)\Lambda\left(V^{-1}V\right)\dots\left(V^{-1}V\right)\Lambda V^{-1} \\ &= V\Lambda^a V^{-1}. \end{aligned}$$

Matrix exponentials may be computed in a similar way, using exponents of the eigenvalues:

$$\begin{aligned} \exp(M) &= \sum_{t=0}^{\infty} \frac{M^t}{t!} \\ &= \sum_{t=0}^{\infty} V\frac{\Lambda^t}{t!}V^{-1} \\ &= V\exp(\Lambda)V^{-1} \end{aligned}$$

2.3.2 Stochasticity

A *stochastic matrix* is a square matrix of non-negative values, where each column sums to 1. The latter condition is satisfied for some matrix M iff the vector of ones $\mathbf{1}$ is a left-handed eigenvector of M with eigenvalue 1 (i.e. if $\mathbf{1}^\top M = \mathbf{1}^\top$). It is not difficult to see that the product of two stochastic matrices is itself a stochastic matrix, as

$$\mathbf{1}^\top(XY) = (\mathbf{1}^\top X)Y = \mathbf{1}^\top Y = \mathbf{1}^\top;$$

the non-negativity constraint is satisfied for XY, as each entry of the product is a sum of products of the (non-negative) entries of X and Y, and must therefore be non-negative itself.

We use stochastic matrices to represent linear transformations of probability distributions which respect conservation of probability. If an $n \times n$ stochastic matrix is applied to a column vector of n non-negative elements, then the elements of the result will also be non-negative, and will have the same sum.

2.3.3 Ergodicity

A stochastic matrix M is said to be *ergodic* if there exists some power k_{min} such that for all $k > k_{min}$, every element of M^k is nonzero. When M is interpreted as a transformation on probability distributions (§2.4), this may be specified equivalently in terms of the paths between pairs of states: if there exists k_{min} such that for every $k > k_{min}$, and for pair of states s_i, s_j, there is a nonzero chance of producing s_j from s_i in exactly k transformations.

Ergodic matrices have exactly one eigenvalue equal to one, with all others falling within the unit circle. As a result, there exists a unique fixed point (the right-handed eigenvector corresponding to the eigenvalue 1) of M to which the columns of M^k approach as k tends to infinity. To see this, consider the product $M^k\mathbf{p}(0)$. The vector $\mathbf{p}(0)$ may be written in terms of the eigenvectors (since they form a basis for $\mathbb{R}^n$), so we have

$$\begin{aligned} M^k\mathbf{p}(0) &= M^k\left(c_1\mathbf{v}_1 + \ldots + c_n\mathbf{v}_n\right) \\ &= \left(c_1\lambda_1^k\mathbf{v}_1 + \ldots + c_n\lambda_n^k\mathbf{v}_n\right) \end{aligned}$$

where the λ_i are the eigenvalues, and the $\mathbf{v}_i$ are the corresponding eigenvectors. Without loss of generality, let λ_1 be the (only) eigenvalue equal to one. Then $\lim_{k\to\infty} M^k\mathbf{p}(0) = c_1\mathbf{v}_1$, as all other eigenvalues have modulus less than one and therefore disappear as $k \to \infty$. The eigenvector $\mathbf{v}_1$ is therefore the unique fixed point, subject to normalisation by the constant c_1.

2.4 Markov Models

Let $\mathcal{X}$ be some set. A probability distribution on $\mathcal{X}$ is a function $p : \mathcal{X} \to [0,1]$, such that $\sum_{x\in\mathcal{X}} p(x) = 1$. If $|\mathcal{X}|$ is finite, then p may be described more concisely as a column vector $\mathbf{p}$ of length $|\mathcal{X}|$, such that $p_i = p(x_i)$.

Now, suppose X and X' are random variables with states in $\mathcal{X}$, with X' being the result of applying some Markovian transformation to X. By definition, such a process has the Markov property, i.e. $\mathbb{P}(X')$ depends only on X, and thus $\mathbb{P}(X' = x_i|X = x_j)$ is a constant. We now take the identity

$$\mathbb{P}\left(X' = x_i\right) = \sum_j \mathbb{P}\left(X' = x_i|X = x_j\right)\mathbb{P}\left(X = x_j\right)$$

and rewrite using vector notation for the probability distributions, with symbols $\mathbf{p}$ and $\mathbf{p}'$ for the distributions of X and X' respectively:

$$p'_i = \sum_j M_{ij}p_j.$$

where M is a matrix of constants in $[0, 1]$, such that $M_{ij} = \mathbb{P}(X' = x_i | X = x_j)$. Of course, this is just an element-wise description of matrix multiplication, and thus $\mathbf{p}' = M\mathbf{p}$. Note that from conservation of probability, $\sum_i \mathbb{P}(X' = x_i | X = x_j) = 1$, and thus each column of M sums to one and M is therefore stochastic.

Chapter 3

Markov Models of Search Heuristics

The heuristics we described in §2.2 possess some common features which simplify analysis. If the parameter schedule is considered an external input, then the state of any of these algorithms is determined entirely by the last point visited; the state space of the algorithm is identical to the search space of the problem. Hence each algorithm can be described as a Markov process, albeit subject to parameters (such as the annealing schedule). This unfortunately precludes the use of either history information (e.g. Tabu search) or populations of search agents as in a genetic algorithm, although we later (§6) adapt the model to consider a population-based variant of Simulated Annealing.

In fact, we usually do not model the search space so directly, for probability distributions over the entire search space are, for any nontrivial problem, too large to manipulate. Instead, a *state amalgamation* process (§4.3) is applied to divide the search space into groups of points; these groups are then used as the states for the Markov model. The models described below refer without assumption to a search space which may be either the original search space or some abstraction of it.

3.1 Search Heuristics

The behaviour of a search heuristic at any given iteration will often depend on the number of iterations performed previously, a quantity which we will simply refer to as "time". This is true of both simulated annealing and descent with variable mutation; in both cases the dependency is indirect, via a parameter which itself depends on time.

For a model of a search to be produced, three objects relating to the search space need be supplied:

m - A stochastic matrix describing the connectivity of the space

$\mathbf{p}(0)$ - A vector of the probabilities of starting at each point in the space

$\mathbf{c}$ - A vector of the costs of each point in the space

As discussed earlier (§2.1.1), the connectivity (i.e. the neighbourhood definition) for a search space is usually derived from the definition of the cost function in such a way as to "help" a heuristic by providing some systematic non-uniformity in the cost distribution of the neighbourhoods, which the heuristic may then exploit to guide the search. In some cases, the neighbourhood definition may be designed so as to work well with a specific search heuristic; otherwise the usual choice is the simplest (or perhaps least arbitrary) definition which produces a connected search space. The matrix m is assumed to be normalised so that the columns sum to unity (i.e. m is stochastic); since m represents the transformation caused by mutation to a random neighbour, we will often refer to m as the *mutation matrix.*

The initial distribution, $\mathbf{p}(0)$, gives the probability of the search starting at any given state. One usually assumes that all points in the search space are equally likely to be chosen for a starting point, so for a direct model, we usually have $\mathbf{p}(0) \propto \mathbf{1}$, while the probabilities for the states of an indirect search space model are usually proportional to the size (i.e. the number of search points represented) of the states. Non-uniform initial distributions may be appropriate when external information suggests particular points of interest; if there were some means of producing an approximation to the global minimum, it would not be unreasonable to favour particular starting points.

Where state amalgamation is used, the entries of the cost vector $\mathbf{c}$ usually[1] equal the costs of the search points represented by each state. A state is a global minimum if the corresponding entry in the cost vector is the minimum entry; we do not require this to be unique.

In §3.2, we will use the transition matrices for these search algorithms to produce expressions for the average cost of a search. For simulated annealing and variable mutation, these costs will be minimised (over the schedule parameters β) using the Scaled Conjugate Gradient algorithm (we use the implementation from Nabney (2001)). Since SCG is gradient based, we must also compute the gradient of the cost function with respect to the schedule parameters; due to the linearity of the model this is not too difficult.

The cost gradient is a vector of the derivatives taken with respect to each element of the schedule. Since each schedule parameter $\beta(t)$ only affects only the corresponding

[1]Although this is true for our experiments, one could easily divide the search space in such a way that the groups contain a mix of costs, in which case the average, or perhaps the minimum or maximum could reasonably be taken as the cost for the state. As with the neighbourhood definition, the entries of cost vector *could* be adjusted to reflect some externally obtained knowledge which would otherwise be unavailable to the heuristic.

transition matrix $w(t)$ (i.e. $t \neq t' \implies \frac{\partial w(t)}{\partial \beta(t')} = 0$), there is only one derivative to compute for each transition matrix. Additionally, the elements of the transition matrices for both simulated annealing and variable mutation have derivatives which can be expressed in terms of (and partially computed from) the elements themselves.

3.1.1 Descent

Although we will make very little use of basic descent, we describe a model here for completeness, and as a foundation for models of other heuristics. For each iteration of the algorithm (§2.2.1), a random neighbour is chosen, and is adopted as the new point if it has cost no worse than that of the current point. We define an *acceptance matrix* A to encode information about which transitions are permitted; the element A_{ji} gives the probability that a proposed transition from state $i \to j$ will be accepted. For descent, we simply have $A_{ji} = [c_j \leq c_i]$.

For any $i \neq j$, the transition probability is the product of that of selecting j, and that of accepting j; we write $\mathbb{P}(i \to j) = (m \otimes A)_{ji}$. Note that we use the symbol '$\otimes$' to denote *element-wise* matrix multiplication, so that $(X \otimes Y)_{ij} = X_{ij} Y_{ij}$.

The probability of a self-transition (i.e. from $i \to i$) is slightly more complicated, as this can come about not only through selecting the same state, but also through the rejection of any other proposed transition. Let w be the transition matrix for a single iteration of descent. We know that $w_{ji} = (m \otimes A)_{ji}$ except when $i = j$. Since w must be a stochastic matrix, we can compute the correct values for the diagonal terms by choosing them such that the columns sum to 1. We therefore have

$$w = m \otimes A + \operatorname{diag}\left(\mathbf{1}^\top - \operatorname{sum}(m \otimes A)\right)$$

where sum and diag are given by

$$\begin{aligned} \operatorname{sum}(M) &= \mathbf{1}^\top M \\ (\operatorname{diag}(\mathbf{v}))_{ij} &= [i = j]\, v_i. \end{aligned}$$

3.1.2 Simulated Annealing

Simulated annealing can be described in a very similar way to descent, the only difference being the contents of the acceptance matrix. From our definition (§2.2.3), the acceptance probabilities are given by

$$A(t)_{ji} = \begin{cases} 1 & c_j \leq c_i \\ e^{\beta(t)(c_i - c_j)} & c_j \geq c_i \end{cases}$$

where β is the annealing schedule. Unlike descent, the transition matrix for simulated annealing depends on the time t.

Omitting the parameter t for brevity, the derivative (of $w(t)$ w.r.t. $\beta(t)$) is given by

$$\begin{aligned}\frac{\partial w}{\partial \beta} &= \frac{\partial}{\partial \beta}\left(m \otimes A + \operatorname{diag}\left(1^{\top} - \operatorname{sum}(m \otimes A)\right)\right)\\ &= m \otimes \frac{\partial A}{\partial \beta} - \operatorname{diag}\left(\operatorname{sum}\left(m \otimes \frac{\partial A}{\partial \beta}\right)\right)\end{aligned}$$

where

$$\begin{aligned}\left(\frac{\partial A}{\partial \beta}\right)_{ji} &= \begin{cases} 0 & c_j \leq c_i \\ (c_i - c_j)\, e^{\beta(t)(c_i - c_j)} & c_j \geq c_i \end{cases}\\ &= \min(0, (c_i - c_j))\, A_{ji}.\end{aligned}$$

Note that the values $\min(0, (c_i - c_j))$ are independent of β, and thus may be precomputed.

The transition matrix for simulated annealing at a given temperature is ergodic (so long as the matrix m produces a connected graph of states[2]), and therefore has a unique fixed point, which is Boltzmann distribution over the costs of the states, weighted by the state sizes (§A.1). For very high temperatures, the Boltzmann parameter is small, so the distribution is near-uniform; as the temperature decreases, the probability mass becomes concentrated around the minima, with the fraction assigned to the global minimum slowly approaching one.

An attractive property of the heuristic is the idea that convergence on the global optimum can be achieved with arbitrarily high certainty, so long as the temperature decreases sufficiently slowly to allow the equilibrium to be reached at each stage (Hajek, 1988); the global minimum can be reached with certainty only as the length of the annealing schedule approaches infinity.

3.1.3 Descent with Variable Mutation

For descent with variable mutation, we replace the mutation matrix m with a matrix $M(t)$, representing the result of performing a Poisson-distributed number of mutations,

[2]Additionally, ergodicity requires that the graph of w be aperiodic. This follows if either m is aperiodic, or if there is nonzero probability of a self-transition in w. The latter is almost always satisfied, requiring only that β^{-1} be nonzero and that the search space contain at least two distinct costs.

with mean determined by $\beta(t)$. We compute M by averaging over the number of mutations:

$$\begin{aligned} M &= \sum_{n=0}^{\infty} m^n e^{-\beta} \frac{\beta^n}{n!} \\ &= e^{-\beta} \sum_{n=0}^{\infty} \frac{(m\beta)^n}{n!} \\ &= e^{-\beta} e^{m\beta} \\ &= e^{\beta(m-I)} \end{aligned}$$

Note that the matrix exponential e^{m-I} may be computed efficiently from the eigenvalue decomposition of $(m - I)$. If D is a diagonal matrix containing the eigenvalues of $(m - I)$, and v is matrix of the corresponding eigenvectors, then $e^{m-I} = ve^D v^{-1}$, with e^D being simply the diagonal matrix composed of the exponents of the eigenvalues.

The derivative (again, omitting the parameter t) is given by

$$\frac{\partial w}{\partial \beta} = \frac{\partial M}{\partial \beta} \otimes A - \text{diag}\left(\text{sum}\left(\frac{\partial M}{\partial \beta} \otimes A\right)\right)$$

where

$$\frac{\partial M}{\partial \beta} = (m - I)\, e^{\beta(m-I)}$$

3.2 Cost of a Search Heuristic

Equipped with the above descriptions of search heuristics, we are now in a position to describe the possible paths through the search space taken by a heuristic, as

$$\mathbf{p}(T) = \left(\prod_{t=1}^{T} w(t)\right) \mathbf{p}(0) = w(T) \cdot w(T-1) \ldots w(2) \cdot w(1) \cdot \mathbf{p}(0)$$

with $\mathbf{p}(T)$ giving the probability of being at various points at time T, averaged over all possible histories of points for $t < T$. Note that we are adopting the convention that a product denoted by the '$\prod$' symbol is evaluated as shown above, with each term *pre*-multiplying the product of all previous multiplicands.

We now consider various measures of search performance, each of which calculates a "cost" for a heuristic, based on $\mathbf{p}(T)$. Here, we examine two common definitions of cost (Where-You-Are and Best-So-Far); in §7.2 we also consider a third measure, the average first passage time. The three cost measures are illustrated in fig. 3.1. In all three cases, it is assumed that the only relevant statistics are execution time and solution quality, and thus knowing the solution quality as a function of time is sufficient. Since our heuristics are permitted only one cost evaluation per iteration, the "time" is effectively the number of evaluations. We assume that the computational expense of cost evaluation dominates that of all other operations performed by a heuristic.

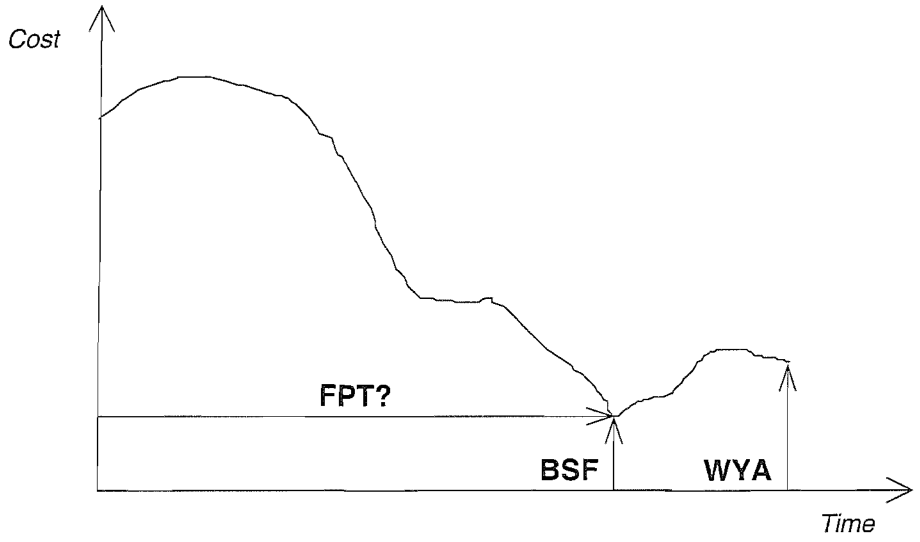

FIGURE 3.1: Cost of the points visited by a search heuristic vs iteration number (time). The three measures of search cost indicated are "Where-You-Are", "Best-So-Far" and First Passage Time (defined later in §7.2). The FPT label is conditional on the cost indicated being the global minimum, otherwise FPT is unknown (and potentially undefined).

We also assume that each cost evaluation takes approximately equal time; in reality, the evaluation time may vary considerably. For many problems, the cost function is sufficiently structured that the cost of a search point may be partially computed from that of a neighbour; such time-saving strategies are assisted by the tendency of most search heuristics to make decisions based only on cost *differences* between search points. In reality the computation time for a cost evaluation may even vary as a function of the set of points previously visited by the heuristic.

3.2.1 Where-You-Are (WYA) Cost

The "Where-You-Are cost" of a heuristic is simply the average cost of the last point visited. If T iterations are performed, this is simply given by $C_{wya} = \mathbf{c}^{\top}\mathbf{p}(T)$. The derivative of C_{wya} with respect to the schedule $\beta(t)$ is given by

$$\frac{\partial C_{wya}}{\partial \beta(t)} = \mathbf{c}^{\top}\left(\prod_{i=t+1}^{T} w(i)\right)\frac{\partial w(t)}{\partial \beta(t)}\left(\prod_{i=1}^{t} w(i)\right)\mathbf{p}(0).$$

WYA cost has the advantage that both the cost and the derivative are quite straightforward to compute. However, it is not directly relevant to the solution of real problems;

one would expect any practical implementation of a heuristic to record any particularly good solutions, and output these in preference to whichever solution happened to be the last visited. This behaviour can of course be modelled directly as part of the heuristic, but doing so effectively *squares* the number of possible states. A much more practical alternative is to measure BSF cost.

3.2.2 Best-So-Far (BSF) Cost

The BSF cost represents the cost of the best state reached at any iteration of the algorithm; the final state is of no extra importance, as all visited states are considered. The implementation is more complex than WYA; we use a method introduced by Boese and Kahng (1994), where the transition matrices are modified to "absorb" particular fitness values.

We compute the average BSF cost by summing over the possible costs. Let α be an ordered list of the k distinct costs in $\mathbf{c}$, such that

$$\begin{aligned} \alpha_i = \alpha_j &\iff i = j \\ \alpha_i < \alpha_j &\iff i < j \\ \forall i, &\quad c_i \in \alpha \\ \forall a \in \alpha, &\quad \exists j \quad s.t. \quad c_j = a \end{aligned}$$

then the average BSF cost is given by

$$C_{bsf} = \sum_{a \in \alpha} \mathbb{P}\left(b(T) = a\right)$$

where $b(T)$ is a random variable representing the minimum of all the costs up to time T. Note that $b(T)$ is not simply the minimum of $\mathbf{c}^\top \mathbf{p}(t)$ over all $t \leq T$; the average BSF cost is not the same as the BSF average cost.

We define $\mathbf{m}_a$ to be a "mask vector" picking out all costs of a or better; thus $(m_a)_i = [c_i \leq a]$. We now replace the transition matrices $w(t)$ with a set of matrices $d_a(t)$, given by

$$d_a(t) = w(t) + \mathbf{1}\mathbf{m}_a^\top \otimes (I - w(t)) \,.$$

The matrix $d_a(t)$ is identical to $w(t)$, except in the columns selected by $\mathbf{m}(a)$, which correspond to the states of cost a or better; these columns are replaced with the corresponding columns of the identity matrix. The effect is that once a state of cost a or better is entered, the search will be trapped there. When the matrices are applied to simulate a search, the sum of the probabilities from the "trapped" costs gives the probability that a cost of a or better was reached during the search. Thus the cumulative distribution of $b(T)$ is given by

$$\mathbb{P}\left(b(T) \leq a\right) = \mathbf{m}_a^\top \left(\prod_{t=1}^{T} d_a(t)\right) \mathbf{p}_0$$

with the probabilities for individual values being computed by comparing adjacent terms (with the boundary case given by $\mathbb{P}\left(b(T) = a_k\right) = 1$).

The derivative of C_{bsf} can be computed in a similar manner to that used for C_{wya}, with the derivatives of d_a being given by

$$\frac{\partial d_a(t')}{\partial \beta(t)} = \left[t = t'\right] \left(\frac{\partial w(t)}{\partial \beta(t)} - \mathbf{1}\mathbf{m}_a^\top \otimes \frac{\partial w(t)}{\partial \beta(t)}\right),$$

which is again similar to $w(t)$, but with the columns for costs of a or better replaced with zeros.

3.2.3 Other Measures of Cost

Other measures of search cost are of course possible. The average first passage time, defined in detail in §7.2, is the average time taken to reach a global minimum (one could similarly measure the average time taken to reach a state with cost below some threshold, as in Cohn and Fielding (1999)). The probability of reaching (or finishing in - Franz and Hoffmann (2002)) the global optimum is also a candidate for study.

We assumed earlier (§3.2) that the number of cost evaluations and the solution quality are the only pertinent statistics for evaluating the performance of the search; in reality, the time taken for each cost evaluation may be non-uniform (or the evaluation may require consumption of some resource other than time), and thus the cost of the search might depend on *which* points were evaluated.

There exist problem domains where the cost of a point is the output of some noisy process controlled by a "certainty" parameter; exact cost evaluation may be very expensive, but an approximation may be calculated with far less resource consumption. A heuristic able to use partial cost evaluations (perhaps requiring only an upper or lower bound on the cost) should obviously not be judged on the number of evaluations taken. In such situations, the parameter schedule may need to be a continuous function of "time" (or whatever resource is consumed).

3.3 Coarse-Grained Schedules

Producing an optimal schedule involves minimising cost (e.g. BSF or WYA) over the parameter set. Thus the task of producing an optimal schedule of length T is itself a

T-dimensional continuous optimisation problem. Although the availability of gradient information aids the search, the optimisation becomes increasingly difficult as T grows, both in terms of running time and maintaining numerical accuracy. One approach for circumventing this problem is to use parameter schedules which are "coarse-grained" in time; that is, schedules for which several adjacent search iterations are controlled by a single parameter.

In using such schedules, our intent is mostly to reduce the computational effort required, by reducing the dimensionality of the problem. Although we shall accomplish this by controlling equal-size "blocks" of iterations using a single parameter, there do exist alternative ways to describe the schedule. For instance, where we assume a constant parameter for each block; an alternative model could use a linearly varying parameter, or perhaps variable-length blocks.

Care must be taken however to avoid a parameterisation which is preconditioned towards describing a particular type of schedule. Our coarse-graining is not immune to such effects: the implicit assumption that the schedules are locally smooth is not universally true (optimal BSF schedules for graph bisection problems featured periodically spiking temperatures (Boese and Kahng, 1994)). However, any artifacts produced by our intuitively simple coarse-graining scheme are likely to be noticed more readily.

If $\beta(t)$ is a coarse-grained schedule of length T, with *block size* b, then we require that the elements of $\beta(t)$ are described entirely by a schedule $\alpha(t)$ with length T/b, via the relationship

$$\beta(t) = \alpha\left(\lceil t/b \rceil\right).$$

Chapter 4

Test Problems

We shall employ several types of problem as test applications for our heuristics, all of which are specified in terms of binary strings; the search space for each is the set of all binary strings with some fixed length L, i.e. the space $\{0,1\}^L$.

It is common to define the neighbourhood around a search point to be all those points which can be reached through a minimal modification to that point. Thus it is a natural choice to use a "single-bit mutation" (by which we mean the inversion of a single bit of a bit-string) as our minimal modification. The least number of such mutations required to transform one string to another is known as the *Hamming distance*, defined formally as

$$H : \{0,1\}^L \times \{0,1\}^L \to [0,L] \cap \mathbb{Z}$$
$$H(x,y) = \sum_{i=1}^{L} [x_i \neq y_i]$$

thus the set of neighbours for a given bitstring x is

$$n(x) = \{y : H(x,y) = 1\}.$$

For sake of convenience, we reserve the term "neighbourhood" for situations where the point itself is included. We would say that x is inside the neighbourhood of x, but not that x is a neighbour of itself, and thus x does not lie within its own neighbour set. Each bitstring will therefore have exactly L neighbours.

Problems based on binary strings have been chosen for convenience of implementation; no part of our analysis assumes a particular connectivity of the search space.

4.1 Hurdle Problem

The hurdle problem was introduced in Prügel-Bennett (2004b) as an example of a landscape on which a population-based algorithm, through use of recombination, gains a significant advantage over a purely local search heuristic. The landscape features a series of deceptive "hurdles" which must be overcome in order to reach the global minimum.

The cost function for the kth order hurdle problem is defined to be

$$c(x) = \lceil H(x,g)/k \rceil - (H(x,g) \mod k)/k$$

where g is some arbitrarily chosen "goal string". Since the details of representation are hidden from search heuristics, all possible choices for the goal string produce identical problems; thus we choose that the goal string be the string containing only zeros, which we shall denote here by 0.

Note that the cost depends *only* on the distance from the goal string; the cost is essentially a linear function of this distance, but with a spike in cost (a "hurdle") every k steps. We use a hurdle problem of order 2, where the cost can be described more simply by

$$c(x) = H(x,0)/2 + [H(x,0) \text{ is odd}];$$

the relationship between cost and Hamming distance is shown in (fig. 4.1).

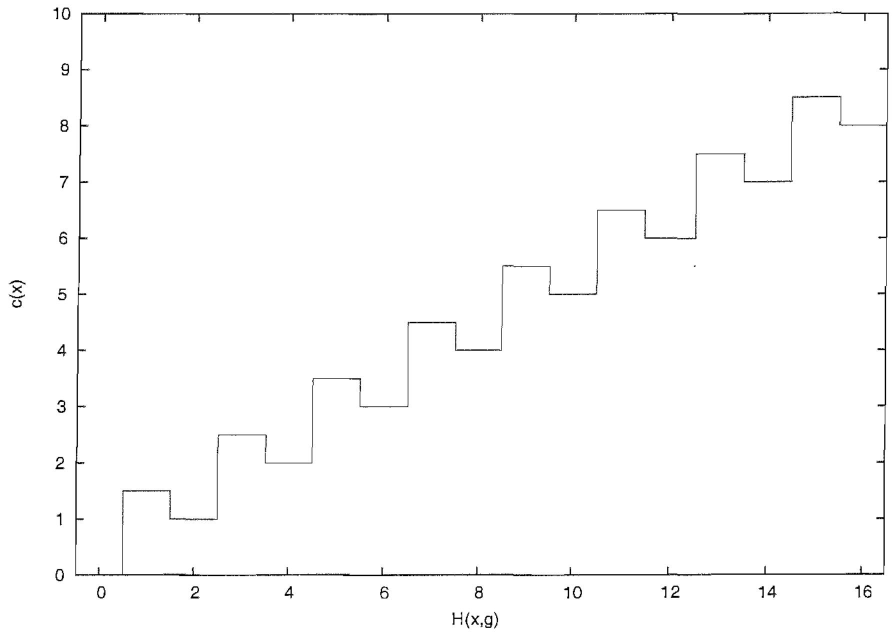

FIGURE 4.1: Hurdle function cost vs hamming distance from goal string, for a 16-bit hurdle problem

With our chosen neighbourhood function, the connectivity of the search space (the equivalent to that of the vertices of the L-dimensional hypercube) forms a bipartite graph, with the two vertex sets being the sets of points of odd and even distance from the goal string. Every even-distance point has only odd-distance neighbours, all of which will be of higher cost, and so every even-distance point is a local minimum. (In fact, no cost function can be defined on this space to produce a higher number of minima: §A.2).

The abundance of local minima is thought to make the hurdle problem challenging for search heuristics. In particular, descent performs very poorly, as it is able to take at most one step before terminating, and even that one step is (at least for first-improvement descent) an indiscriminate selection of a random neighbour.

The search space is of size 2^L; for any L of non-trivial size, this space is too large to use as the state space for a Markov model. Fortunately, the high degree of symmetry present in the cost function allows the state space to be reduced without loss of (significant) information.

For any search point, the only pieces of information directly observable to a heuristic are the cost of that point, and the set of neighbours surrounding that point. However, the cost of a point uniquely identifies the hamming distance, and thus also determines the cost distribution of the neighbours. Thus instead of modelling the path of a heuristic through the search space, we can model the path through *cost space*; the behaviour of the cost at each iteration is itself a markov process, and can be modelled independently of the actual search points visited. Thus a Markov model for the L-bit hurdle problem need have only $L+1$ states.

The effect of applying a random single-bit mutation to a bitstring of length L will be to increase or decrease the number of ones (and thus the hamming distance from the goal string) by one, with probabilities depending on the number of ones and zeros already present. Thus the probability of a string with i ones mutating to have j ones is given by

$$\mathbb{P}(i \to j) = \begin{cases} i/L & \text{if } \; j = i-1 \\ 1 - i/L & \text{if } \; j = i+1 \\ 0 & \text{otherwise} \end{cases} .$$

The initial probability distributions, in terms of the hamming distances from the goal string, are given by a binomial distribution (L trials of probability $1/2$), and thus $(\mathbf{p}(0))_i = 2^{-i}\binom{L}{i}$.

4.2 Real Problems

The hurdle problem is a rather artificial construct, designed with ease of analysis in mind. The local minima, despite being plentiful in number, all have very similar structure; the uphill jumps required to escape them are all of identical size. Real problems have a much more irregular structure, with minima consisting of nontrivial subsets of the search space.

For this reason, we shall also test our heuristics against a set of "real" problems. The Max-SAT, binary perceptron and spin glass problems are all known to be NP-hard (Garey and Johnson, 1979)(whereas a solution to the hurdle problem (with unknown goal string) can be found in $O(L)$ operations). The problem instances we use will be mostly 20-variable problems (search space of approximately a million states). Although considered trivially-sized by those specialising in producing such problems, they are rather large from the point of view of one who wishes to model the behaviour of a heuristic on a problem.

4.2.1 Max-SAT

The boolean satisfiability problem "SAT" was the first problem for which NP-completeness was proved; (Cook, 1971) demonstrated that any NP-complete problem could be reduced to an instance of a SAT problem in polynomial time. Thus far, no polynomial-time solution has been found for SAT, with most of the best approaches (Hansen and Jaumard, 1990) being variations on Davis and Putnam (1960).

A SAT problem consists of a set of *clauses* defined over L boolean variables, which we represent using a bit-string of length L. Each clause is the logical disjunction of a series of k terms, each of which is a variable or its negation; the problem is then said to be a k-SAT problem. For instance, the expression $(x_{30} \vee x_7 \vee \neg x_{19} \vee x_{42})$ is a valid clause for a 4-SAT problem. The difficulty of a randomly generated SAT problem is known to exhibit "phase transition" behaviour when the ratio of clauses to variables reaches a critical point (the whereabouts of which depends on k); the likelihood of a randomly-generated set of clauses being satisfiable drops very sharply at this point (Mitchell et al., 1992; Monasson et al., 1999).

A Max-k-SAT problem has a similar form, but the goal is to find an assignment of the variables which satisfies the greatest possible number of clauses, with the cost of a bit-string given by the number of unsatisfied clauses. Unlike SAT, where the most difficult problems are those around the phase transition (Braunstein et al., 2005), Max-SAT remains difficult when a large number of clauses are used. We mostly use Max-3-SAT problems with 20 variables, and between 120 and 160 clauses. The variables appearing in each clause are chosen randomly and independently; no attempt is made to prevent the occurrence of degenerate clauses such as $(x_{11} \vee \neg x_{11} \vee x_6)$.

4.2.2 Binary Perceptron

The binary perceptron (Krauth and Mézard, 1989) is one of the simplest possible linear classifier systems, with all inputs and weights taking only values in $\{-1, +1\}$. The task of determining the weight vector for a single perceptron to best classify a given set of patterns is known to be NP-hard in the general case (Pitt and Valiant, 1988). In a manner similar to Max-SAT, the problem undergoes a phase transition when the ratio of the size of the classification set to the number of variables grows large enough, at which point the problem becomes more difficult, owing to the introduction of many false optima. These extra optima result from a "shattering" of the search space (Derrida et al., 1991; Prügel-Bennett, 2004a); as the number of patterns increases, each minimum shrinks, until the discreteness of the search space forces it to become disconnected, creating several distinct minima at the same cost.

An L-variable binary perceptron problem consists of a set of data points $\mathcal{V}$, each a vector of length L, containing elements chosen from $\{-1, +1\}$. Each vector $\mathbf{v} \in \mathcal{V}$ is assigned a label $l(\mathbf{v})$, again chosen from $\{-1, +1\}$. A bit-string x of length L is interpreted as a vector $\mathbf{x}$ of this form, and is said to correctly classify a data point $\mathbf{v}$ if the sign of $\mathbf{x}^{\mathsf{T}}\mathbf{v}$ matches that of $l(\mathbf{v})$. The cost of x is then the number of misclassified points, i.e.

$$c(\mathbf{x}) = \sum_{\mathbf{v} \in \mathcal{V}} \left[\mathbf{x}^{\mathsf{T}} \mathbf{v} l(\mathbf{v}) < 0 \right].$$

The data points can be thought of as representing vertices of the L-cube with edge length 2, centred on the origin. The bit-string then represents the normal to a hyperplane dividing the cube (passing through the origin); the data points may be assigned a label $l'(\mathbf{v}) \in \{-1, +1\}$ according to which half of the cube they fall into. The solution of lowest cost is the one which minimises the number of data points for which $l'(\mathbf{v}) \neq l(\mathbf{v})$. We use binary perceptron problems where the data points and the labels are randomly chosen.

4.2.3 Spin Glass

Spin glass models (Mézard et al., 1987) describe the magnetic spins of a disordered collection of atoms under an external magnetic field of varying strength. The weaker the external field, the more the spins are dominated by local influences (either positive or negative correlation) due to the physical arrangement of neighbouring atoms. When these influences are randomly chosen, they frequently contradict one another, producing a "frustrated" system. Models of frustrated spin glasses have been studied both for lattice-like structures (Edwards and Anderson, 1975) with neighbourhood-only influences, and for the generalised case with arbitrary correlations (Sherrington and Kirkpatrick, 1975; Parisi, 1980). The task of determining the ground state (i.e.

the minimum-cost configuration when no external field is applied) is known to be NP-complete.

A bit-string is used to represent the spins of a set of atoms; the cost of a string can be thought of as a sum of correlations between the bits within the string. As with the binary perceptron problem, a bit-string is interpreted as a vector in $\{-1,+1\}^L$. A random $L \times L$ correlation matrix J is created, with $J_{ij} \in \{-1,+1\}$. The cost of a bit-string $\mathbf{x}$ is then given by

$$c(\mathbf{x}) = \sum_{i<j} J_{ij} x_i x_j.$$

4.3 Reducing State Space for Large Problems

Even for 20-variable binary string problems, the search space is too large to make a direct model practical. A search space of size 2^{20} usually requires a mutation matrix with 2^{40} elements, which is beyond the storage (let alone manipulation) capacity of most current computer hardware[1]. Unfortunately, most "hard" problems do not exhibit the kind of obvious symmetry that we were able to exploit with the hurdle problem. There are, however, alternative ways to construct a reasonably-sized model for a problem; we shall use a *barrier tree* model, as in Hallam and Prügel-Bennett (2005a,b); Hallam (2006).

Note that we are constructing barrier trees as models of problems, rather than defining problems by randomly creating barrier trees; the intention is that the trees produced capture the structure of the underlying problems.

This section describes the work carried out by Jonathan Hallam, who also generated the barrier tree data (state sizes, costs and transition probabilities) used for all our work with the Max-SAT, binary perceptron and spin-glass problems.

4.3.1 Barrier Trees

A cost landscape may be described as the disjoint union of a number of level-connected sets (§2.1.2). A neighbourhood relation can be described on these sets, such that two level-connected sets $\mathcal{X}$, $\mathcal{Y}$ are neighbours if for some $x \in \mathcal{X}$, $y \in \mathcal{Y}$, x is a neighbour of y. The connectivity between these sets can be represented as a graph, providing an abstraction to the original search space. While this graph may be used directly to model the search space, the number of level-connected sets is typically very large. *Level-accessibility* provides a way to group level-connected sets into equivalence classes.

Let S be a search space, with neighbourhood function $n : S \to P(S)$, and cost function $c : S \to \mathbb{R}$. An element y is said to be *accessible* from an element x if there exists a

[1]The author is quite aware that this statement will inevitably become a source of future amusement.

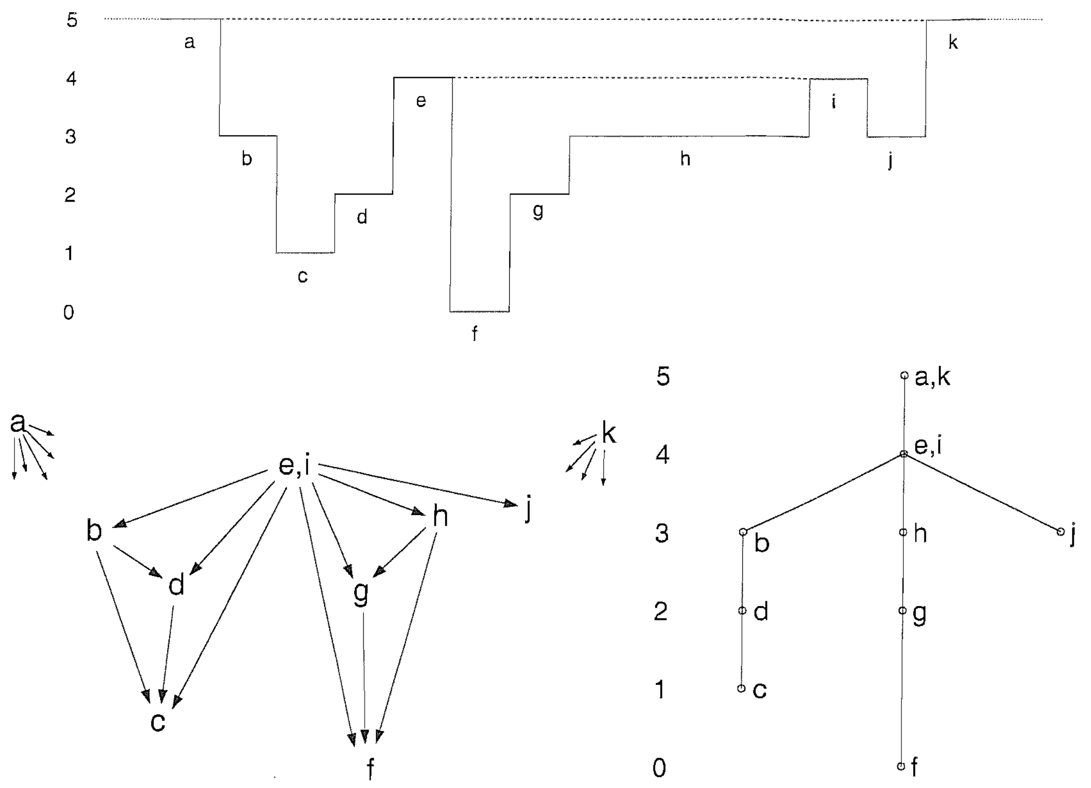

FIGURE 4.2: Barrier tree (lower right) and accessibility digraph for level-connected sets (lower left) for a one-dimensional cost landscape (top). The digraph nodes 'a' and 'k' are each connected to all nodes of the graph (including each other); the edges from these nodes are omitted for clarity. Similarly, 'e' and 'i', which have identical connectivity (and are connected to each other) are shown as a single point.

path between them which does not visit any node of higher cost than that of x; we may express this formally as a predicate on pairs of elements in S:

$$\begin{aligned}\mathrm{acc}(x,y) \iff & \exists L \in \mathbb{Z}^+, \exists p : \mathbb{Z} \to S \\ \text{s.t.} \quad & 1 \le i \le L \implies p(i) \in n(p(i-1)) \\ \text{and} \quad & \forall i \in [0,L]\,, c(p(i)) \le c(x) \\ \text{and} \quad & p(0) = x, p(L) = y\end{aligned}$$

We now define an equivalence relation on S with

$$x \sim y \iff acc(x,y) \wedge (c(x) = c(y))$$

These equivalence classes (known as level-accessible sets) are to be the states for the Markov model. As with level-connected sets, they may be represented as nodes on a graph, with classes S_i and S_j, $c(S_i) > c(S_j)$, connected if S_j is the lowest-cost state to satisfy $\exists x \in S_i, \exists y \in S_j, acc(x,y)$[2]. In fact, this graph is the same as that of the level-connected sets, but with nodes corresponding to level-accessible pairs of level-connected sets identified.

Since each state may only be connected to at most one state of higher cost, no cycles are possible, and thus we have a tree. Although the structure of the tree is not of direct use to the Markov model, it has utility as a tool for visualising the structure of local minima and saddle points within the search space (§4.4). Each leaf node of the tree represents a minimum, while non-leaf nodes are saddle points.

Fig. 4.2 illustrates the relationship between the graphs of level-connected and level-accessible sets. For the one-dimensional cost landscape shown, there are almost as many level-accessible sets as there are level-connected sets; this is atypical of actual cost landscapes, which tend to have a much higher ratio of states to distinct costs, and greater connectivity between those states due to higher dimensionality. A barrier tree for a Max-SAT problem is shown in fig. 4.3.

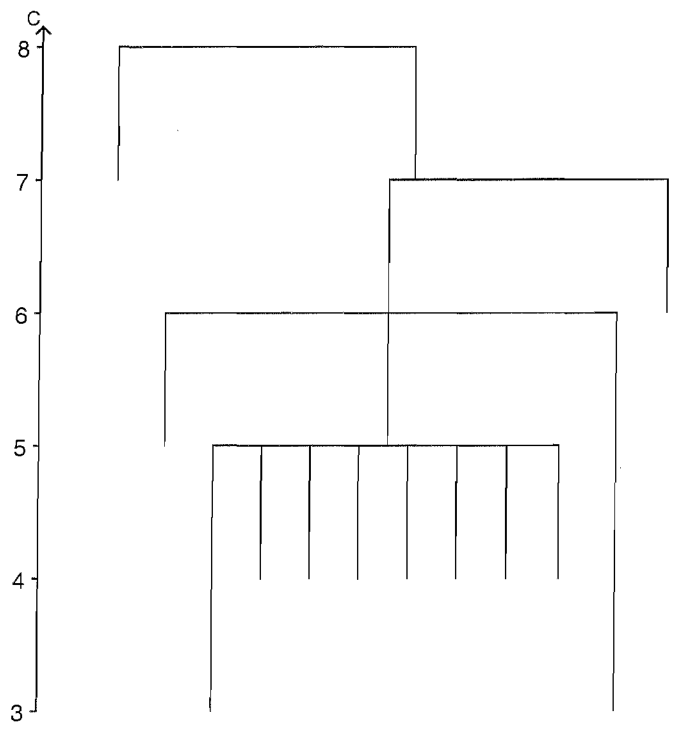

FIGURE 4.3: An example of a barrier tree for a 20-variable Max-SAT problem. Only the area of the tree containing the global minimum is shown.

We note that the 16-bit hurdle problem studied earlier has 2^{15} saddle points (all strings of odd distance from the goal string). However, all saddle points at a given cost are

[2]Note that if accessibility holds for any one pair from $S_i \times S_j$, then it holds for all such pairs. If there exists such a state, it is guaranteed to be unique; otherwise there would be two equal-cost states S_j, S_k connected via a lower-cost state S_i, and thus we would have $S_j \equiv S_k$.

accessible from all others. To see this, observe that constructing a path from the goal string to any saddle point with Hamming distance from the goal string increasing at every step will produce a path with maximum cost at the saddle point; any two equal-cost saddle points may be connected through the concatenation of two such paths. Thus there are only 8 non-leaf nodes in the barrier tree for the 16-bit hurdle problem, corresponding to the tops of the "hurdles" in the hurdle function (fig. 4.1). However, each local minimum is a separate state, so the number of children belonging to each of these saddle-point nodes becomes exponentially large as the "denser" states (those corresponding to many saddle points of equal cost) are reached; this is illustrated by fig. 4.4.

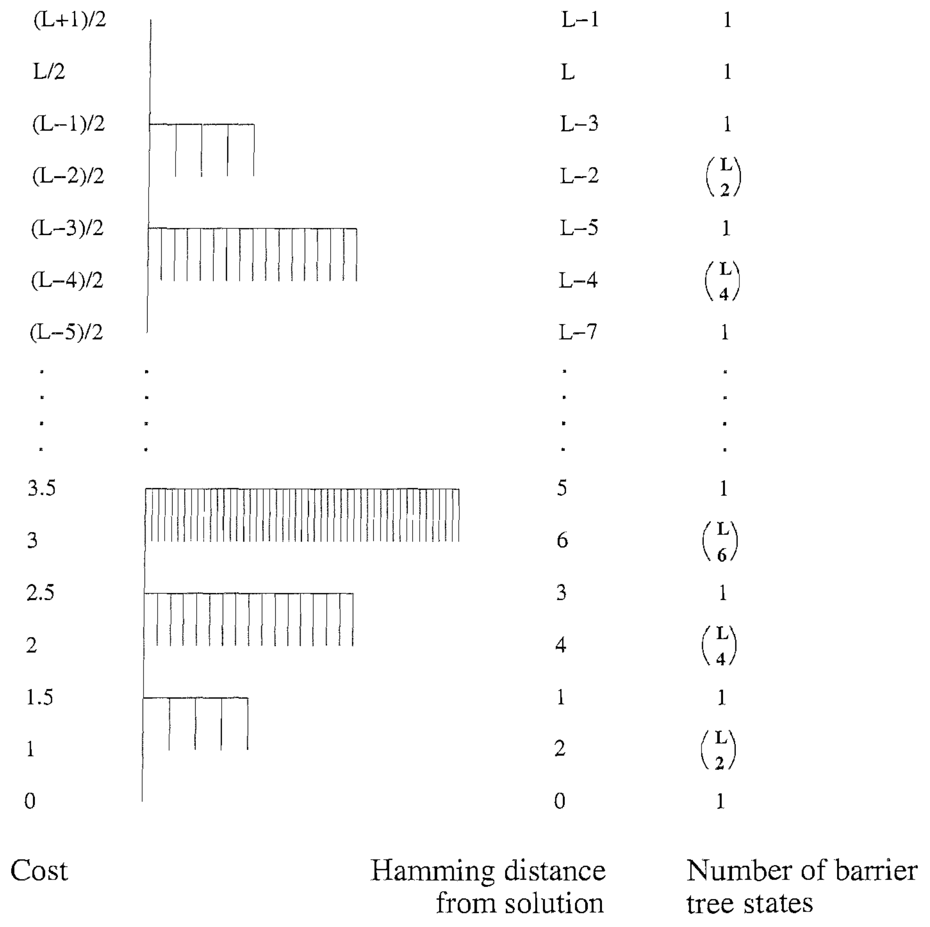

FIGURE 4.4: A barrier tree for an L-bit hurdle problem. Note that every point in the search space is either a local minimum or maximum. Because every local maximum is accessible from all other maxima of the same cost, the vast majority of the nodes correspond to local minima.

4.3.2 Constructing a Markov Model

The barrier tree provides a set of states for our Markov model; however, this is of little value unless we know the mutation probabilities between the states (the matrix m), and the distribution of initial states (the vector $\mathbf{p}(0)$). The latter is produced simply by taking the cardinality of each set, divided by that of the search space. The former is approximated by examining the boundary of each set: enumerating all possible mutations of all elements of a set provides a list of neighbour elements (which *is* permitted to contain duplicate elements). The probability of transition to any other set is then assumed to be equal to the proportion of the neighbour elements which belong to that set.

The probability of a mutation from a *uniformly random* element in state i to *any* element in state j is given exactly by m_{ji}. However, the result of the mutation will not be a random element in state j; the distribution is known to be non-uniform. As a result, the transition probabilities depend on the previous nodes visited (fig. 4.5 gives an example), and we do not have a true Markov process; in assuming that this dependency can be ignored, we are making an approximation.

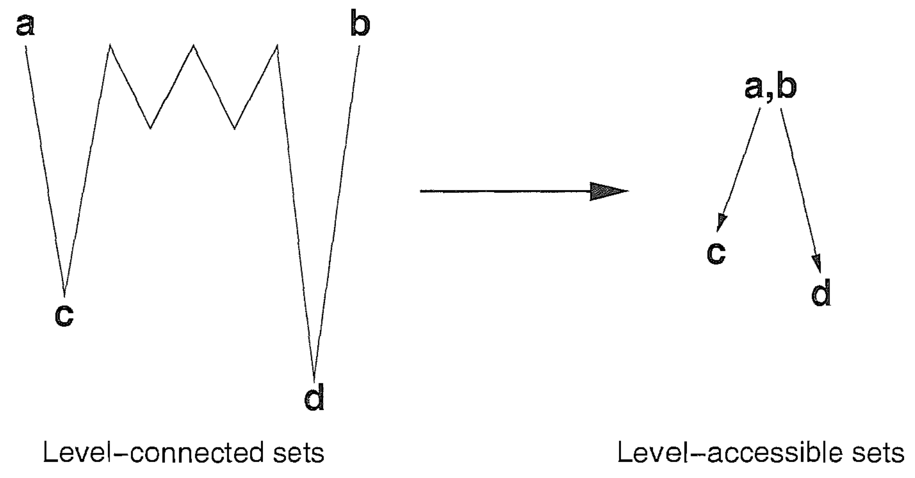

FIGURE 4.5: In the situation shown, the chance of reaching c or d is heavily dependent on which of the higher states a, b one begins in. Information about this correlation is lost however, as a and b form a single level-accessible set.

It is sometimes possible to construct a barrier tree for problems where the state space is too large to search exhaustively. A modified branch and bound algorithm is used to enumerate all points in the search space below some specified cost; in many cases this limited search will uncover the vast majority of local minima (it is of course guaranteed to find the global optimum). The remainder of the search space is then assumed to contain one level-accessible set per distinct cost; the transition probabilities for these extra states may be estimated through random sampling of the search space (Hallam and Prügel-Bennett, 2005b), allowing models of problems as large as 40-variable Max-SAT instances to be constructed with ease.

4.4 Accessibility Graphs for Search Space Visualisation

Our use of (level-accessible) barrier trees is motivated by the belief that, despite losing information through state amalgamation, they preserve much of the structure of the original problem. Ideally, the behaviour of a heuristic on a barrier tree model could be used to predict the behaviour on the real problem. In §5.3 we discover that such predictions tend to be inaccurate, leading us to believe that some information significant to the search is lost in the transformation. Despite this, we will see in this section that the barrier tree abstraction has independent value as a means for visualising the connectivity between various regions (specifically, the level-accessible sets) of the search space.

While we have repeatedly spoken of a "barrier tree model", the part of this model which we have made use of is the decomposition of the search space into a relatively small number of regions (the level-accessible sets). The fact that it is possible to impose a (partial-) ordering upon these regions is somewhat incidental. Furthermore, the arrangement of these regions in the search space does not directly respect this ordering: the adjacency of two sets in the search space (and thus the possibility of moving directly between them) is neither necessary nor sufficient for their representative barrier tree nodes to be connected.

4.4.1 Accessibility Graphs

It is perhaps more appropriate to use a visualisation which emphasises those aspects of the representation upon which our model depends most fundamentally, namely the transition probabilities. In fig. 4.6, we show a graph of the level-accessible sets of a Max-SAT problem. The graph nodes represent those sets, and are labelled by cost. The edges represent transition probabilities, with darker[3] (and thicker) edges corresponding to more probable transitions. Since every edge has a dual[4], we choose to draw only the *downhill* edges, believing these to be the most important to the analysis.

We arrange the nodes on an arc (in fact a spiral) for ease of viewing, with the angle and radius determined by the cost of corresponding state; to have them laid out linearly would make most of the connections between higher states invisible. Nodes with equal cost are aligned radially; those from which the global minima are accessible are part of the main spiral, nodes which lead only to (or are) local minima are separated from these with a little spacing.

[3]The range of shades has been adjusted to exclude those too light to be visible

[4]We actually have a digraph, but with the property that for any edge with nonzero weight, the reverse edge also has nonzero weight. The ratio of the two weights is the same as that of the perimeters of the two sets; while thusly related, they result in subtly different graphs.

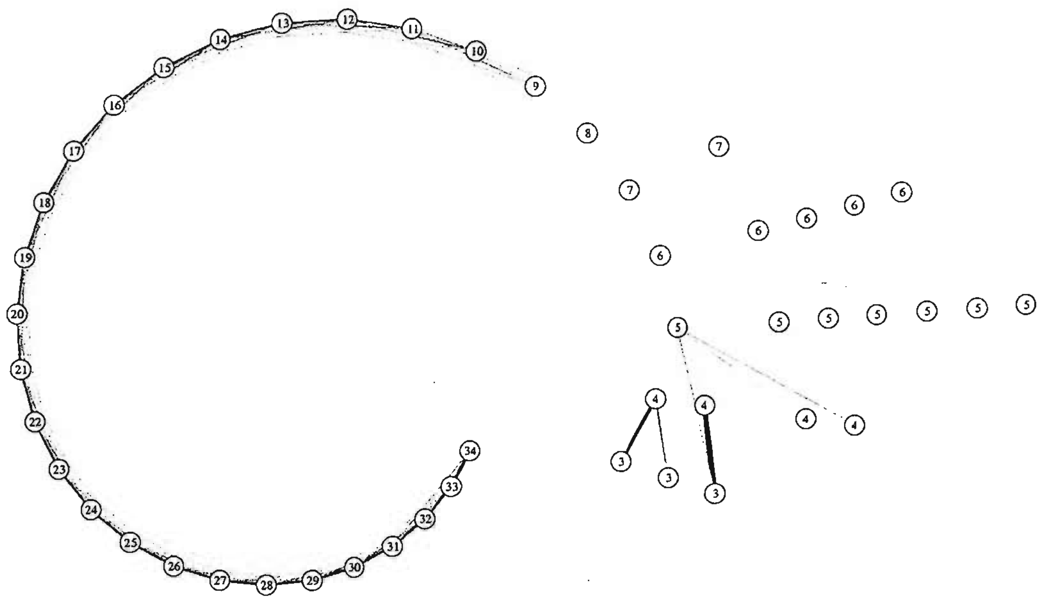

FIGURE 4.6: Barrier tree connectivity for a 20-variable Max-SAT problem.

4.4.2 Connectivity Observations

From fig. 4.6, we see that the minima are concentrated in the lower-cost areas of the search space (although the proportion of the search space associated with each cost could be such that these areas account for most of the search space). This distribution appears to be typical of Max-SAT problems; for each of the higher costs, there is usually just a single level-accessible set. This can also be seen in fig. C.1; the range of costs at which the minima may be found is in each case within the lowest 25% of the cost range. We also see that the range of costs is covered by the non-minimum states without repetition (the number of distinct costs of states is equal to the number of (integer) costs within the cost range); by deduction, the upper 75% of the cost range, being free of local minima, must consist of a single state per cost.

Connections between sets are generally strongest for sets with consecutive costs; however, transitions between nodes with a large cost difference are still possible up to a point. The maximum number of nodes which may be bypassed in this fashion is bounded by (in the case of Max-SAT problems) the maximum number of clauses any variable appears in (i.e. the maximum number of clauses for which the satisfaction may change in response to a change in any one variable). In terms of the cost landscape, we see that the high-cost regions are rough but easily traversable, while the low-cost area is much more fractured. While the global minima are reachable from nodes of non-consecutive cost, this is equally true of the local minima, which are much greater in number. Further, the nodes from which the global minima are accessible appear to be connected with no greater preference than the local minima.

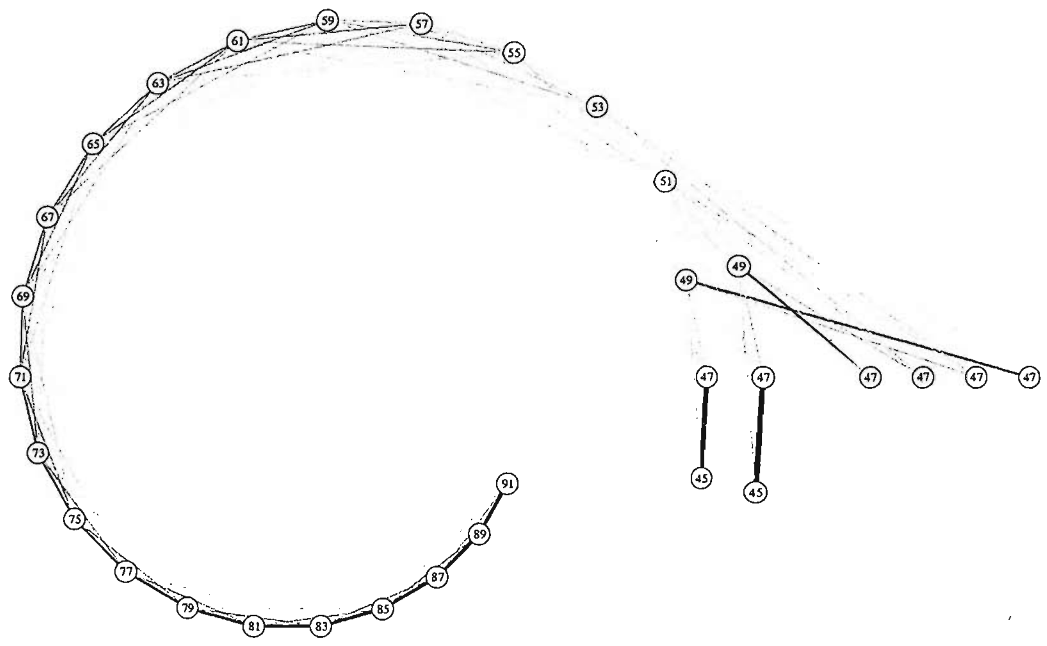

FIGURE 4.7: Barrier tree connectivity for a 17-variable Spin-Glass problem.

The spin glass graphs (fig. 4.7) are not dissimilar to those for the Max-SAT problems. It appears that, at least within our data set, spin glass problems tend to have a simpler landscape than Max-SAT problems of a similar number of variables (the reader should note that the two graphs are for problems of slightly different size); this is corroborated by the lower spin glass first passage times in §7. As with Max-SAT, the minima are concentrated at lower costs; this is consistent with the observations of broken ergodicity in Anderson (1986). The numerical data in fig. C.2 however shows that repeated costs in non-minimum states do occur here, with the number of distinct costs being less than the number of non-minimum states in six of the problems. This could be interpreted as evidence that spin-glass problems are more likely to have minima with non-trivial basins of attraction. It is also possible however that this is due to the smaller problem size (compared to Max-SAT). Increasing the dimension of the problem allows for greater connectivity, but results in a larger number of level-connected sets; since the two effects may grow at different rates, it is difficult to extrapolate to larger problems.

Binary perceptron graphs however have a very different structure. Firstly, local minima are much more common, with problems in fewer than 20 variables frequently having in excess of a hundred level-accessible sets. Larger problems generally had larger numbers of minima (fig. C.3), but the variation at each problem size was quite significant, with figs. 4.8 and 4.9 both representing instances generated using the same parameters.

The distribution of local minima with respect to cost is much more uniform than for the Max-SAT or spin glass problems. In particular, there are local minima at almost every cost, except for the top three or four costs, and the average cost for the minima is close to the centre[5] of this range (fig. C.3. This could be interpreted either as an indication that the minima in this case are distributed more evenly, or alternatively that the distribution of the costs themselves over the landscape is more uniform. Regardless, the result is a landscape which is considerably more difficult for our heuristics (to which only cost information is available) to traverse, as at almost every stage in the search, a transition to a local minimum is indistinguishable from one which leads towards the global optimum.

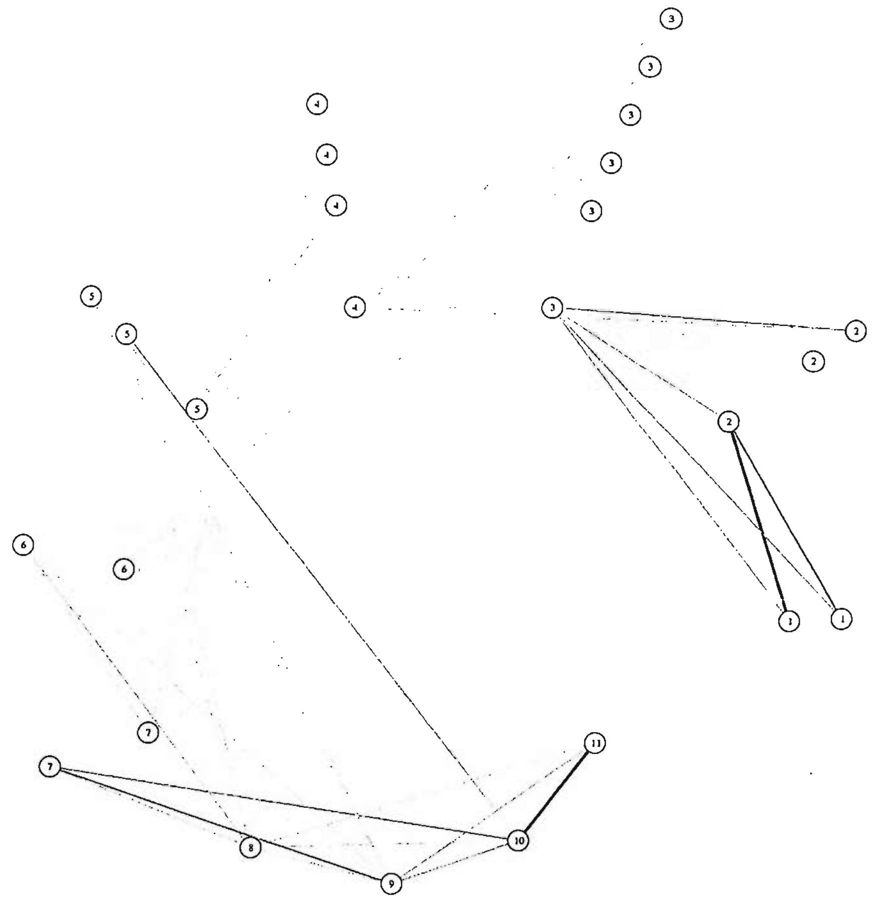

FIGURE 4.8: Barrier tree connectivity for a 12-variable Binary Perceptron problem.

[5]It is interesting to note that the average cost of the minima appears to be determined more by the problem size, and appears to be independent of the minimum cost.

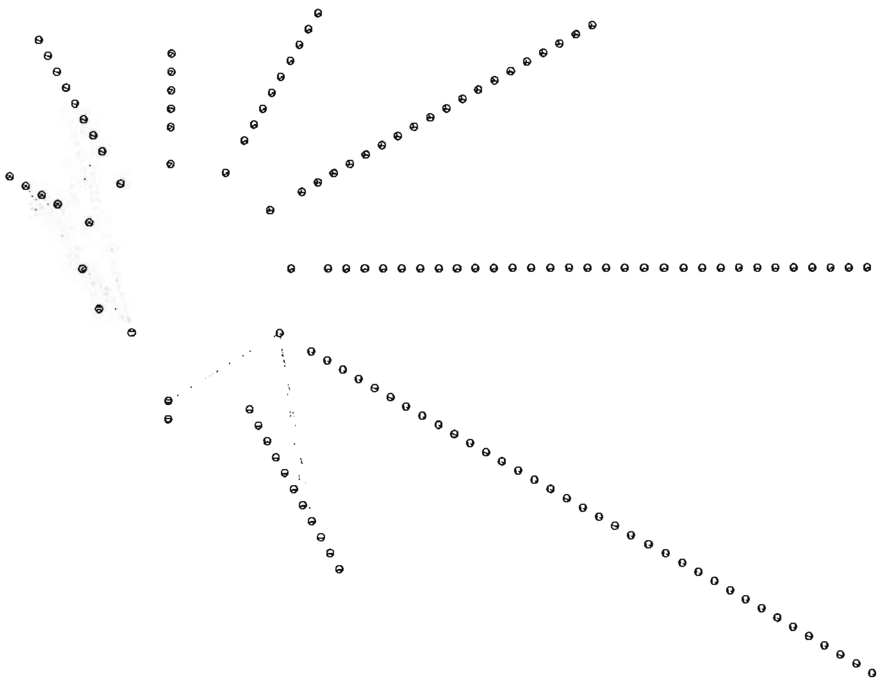

FIGURE 4.9: Barrier tree connectivity for a 12-variable Binary Perceptron problem.

Chapter 5

Optimal Parameter Schedules

In §3, we produced an expression for the average cost of applying a search heuristic on a given problem. In this context, the cost of a search is simply a quantitative measure of performance, derived in some way from the costs of the points visited during the search. In this chapter we produce search parameters chosen in such a way as to minimise the search cost (i.e. maximise performance).

5.1 Optimisation Methodology

The simulated annealing and descent with variable mutation algorithms are similar insofar as their parameters take the form of a "schedule" - a function $\beta(t)$ which defines the value of a parameter at a given iteration t. For both WYA and BSF cost, the cost of the search is defined in terms of the costs of the points reached over some finite, fixed number of iterations T. The parameter schedule must provide a single value for each iteration, and thus the space over which we must optimise is $\mathbb{R}^T$.

Fortunately, the derivatives of search cost with respect to the parameter schedule are not difficult to compute efficiently (see §3), enabling a gradient-based search to be used. The schedules shown in this chapter were optimised using the Scaled Conjugate Gradient method (we use the implementation from Nabney (2001)), starting from a random, log-normally distributed set of values. In most cases, the optimisation was repeated several times, from different starting points; the resulting schedules were not found to differ significantly.

5.1.1 Uniqueness of Solution

There is certainly no guarantee that, for a given search problem, the space of all parameter schedules contains only a single optimum, but repeated optimisation runs (starting from different random schedules) show that we converge on the same schedules (discarding very minor differences which are less significant than the numerical error in the evaluation of schedule cost). It is of course not impossible that the schedules reached are simply local minima of the schedule space, with basins of attraction much larger than the global minimum.

Additionally, the floating-point representation of parameters necessarily discretises both the parameter space and the output of the cost function. The discretising of the cost has the effect of equating near-identical cost values, causing minima to become regions of the space, rather than point values. The discretising of the parameter space itself produces a sampled view of these regions, artifacts of which will cause the minimal regions to become disconnected. While it is very likely that numerous extra minima will be produced in this way, they will of course be distributed around the "true" minimum, and will be of such similar cost that they will be equally acceptable.

5.1.2 Evidence of Convergence

It is quite reasonable to check for some confirmation that the optimised schedules do in fact represent minimum points in the space of all schedules. Computing the gradient of an optimised annealing schedule (fig. 5.1) shows that almost all components of the gradient have magnitude no larger than 10^{-6}; the gradients of non-optimal schedules (e.g. figs. 5.2, 5.3) typically have components at least an order of magnitude larger. For comparison, the actual costs involved are integers in the approximate range $[0, 30]$ for the Max-SAT problems, $[0, 11]$ for the spin-glass problems, and $[10, 90]$ for the binary perceptron problems (details can be found in §C.1). Furthermore, much of the variation in the gradient at different times resembles noise; in short, the schedule is sufficiently close to the minimum that the error in the computation of the cost gradient is comparable to the gradient itself. We also note that the second derivative is strictly positive (another condition we seek when searching for a minimum); the regions where it appears to be zero are actually slightly positive, with values around 5×10^{-8}.

The optimisation over the schedules was performed with a limit of 500 iterations of the Scaled Conjugate Gradient algorithm; increasing this figure produced schedules with very little discernible difference (§C.2.1), as did using a different optimisation algorithm (the Nelder-Mead Simplex Algorithm (Nelder and Mead, 1965)). From this we conclude that the schedule is as close to the optimum (for our model of the problem) as can be reasonably reached using SCG, and is not significantly far from that optimum.

Fig. 5.2 shows a schedule produced from the optimised one by removing the initial low-temperature region, setting all temperatures before the maximum to be equal to the maximum. The associated derivative suggests, as we would expect, that the temperature in that region should be lowered in order to improve the search. Note that the derivative

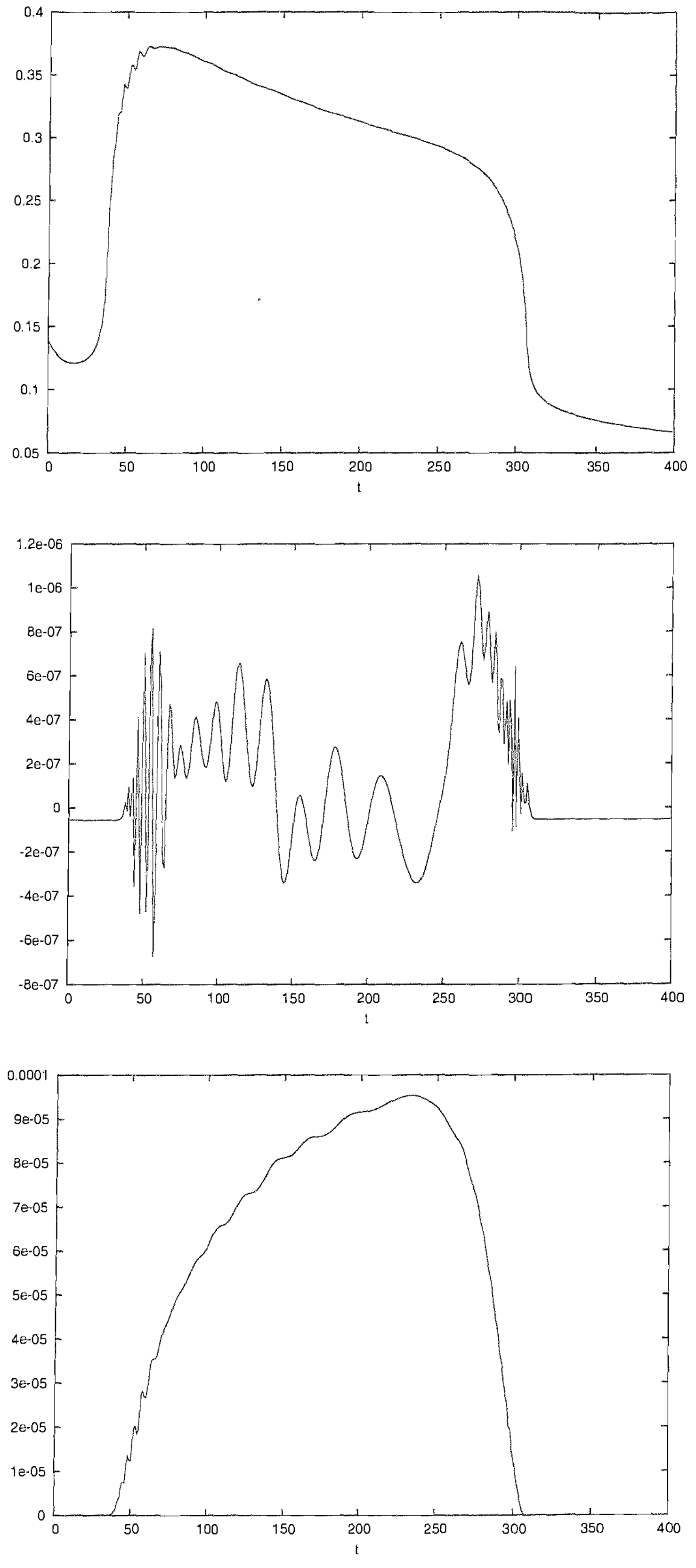

FIGURE 5.1: An optimised schedule for temperature (β^{-1}) against time (top), followed by first and second derivatives of average WYA cost, with respect to the *inverse* temperature β. The second derivatives shown are the non-mixed derivatives, i.e. the main diagonal of the Hessian.

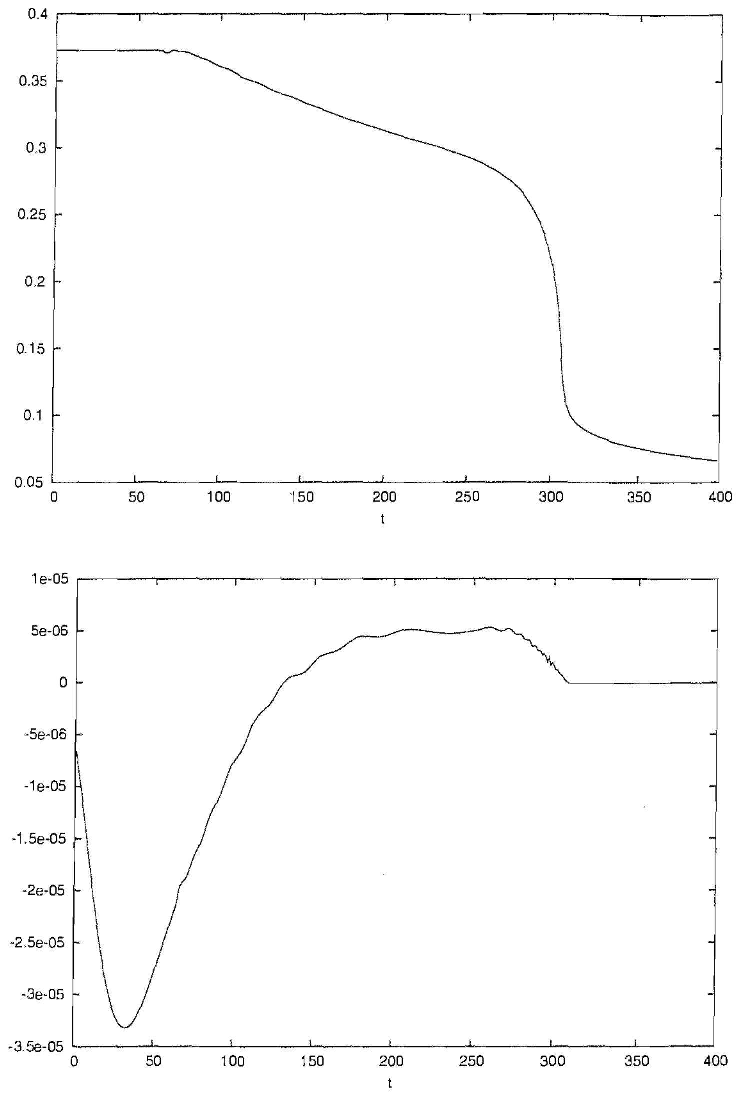

FIGURE 5.2: Temperature and first derivative of cost of a schedule perturbed by removing the "cold start".

is taken with respect to the inverse temperature β; thus a negative derivative indicates that the search cost is lowered by increasing β, thus *lowering* the temperature.

While measuring higher derivatives, we observe an interesting phenomenon: for large enough n, we seem to find that $\frac{\partial^{n+1}}{\partial\beta(t)^{n+1}}C_{wya} \approx -k\frac{\partial^n}{(\partial\beta(t))^n}C_{wya}$, for all t. In other words, each derivative resembles a scaled and reflected version of the previous. For the special case where derivatives are evaluated for a constant-temperature schedule (i.e. one for which $\frac{\mathrm{d}\beta}{\mathrm{d}t} = 0$), it seems that the rule is satisfied precisely, with $k = 1$, as shown in fig. 5.3.

The relationship we observe can be expressed in terms of the matrix products:

$$\forall t, \quad \mathbf{c}^\top \left(\prod_{i=0}^{t-1} w(i)\right)\left(\frac{\partial^{n+1} w(t)}{\partial\beta(t)^{n+1}} + k\frac{\partial^n w(t)}{\partial\beta(t)^n}\right)\left(\prod_{i=t+1}^{T} w(i)\right)\mathbf{p}(0) = 0$$

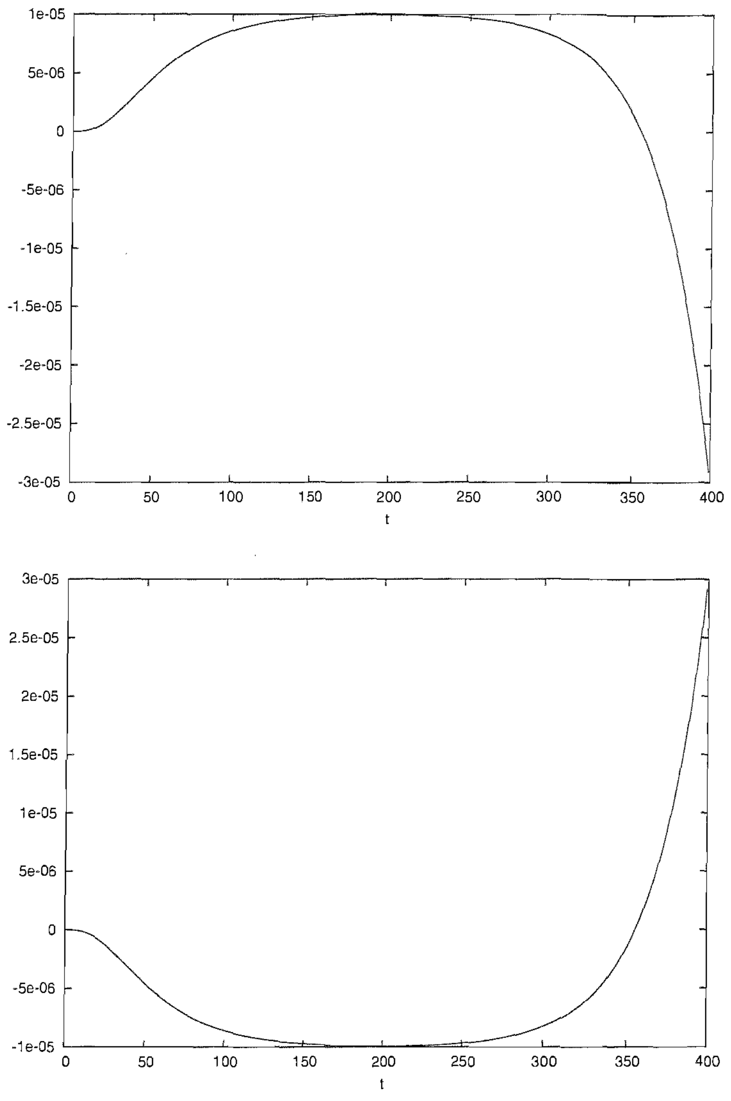

FIGURE 5.3: The first and second derivatives of a constant schedule sum to zero.

For this equation to be satisfied, it is sufficient (but not necessary) for the central parenthesised factor to be a matrix of zeros. For brevity we shall omit the parameter t, and indicate the nth derivative of a quantity by writing n as a parenthesised superscript. Our expression is now

$$w^{(n+1)} + kw^{(n)} = 0.$$

From §3.1.2 we have (for strictly positive n)

$$w^{(n)} = A^{(n)} \otimes m - \text{sum}\left(\text{diag}\left(A^{(n)} \otimes m\right)\right)$$
$$w^{(n)}_{ji} = A^{(n)}_{ji} m_{ji} \; - \; [i = j] \sum_k A^{(n)}_{ki} m_{ki},$$

from which we note that, since every term on the right-hand side has exactly one element of the acceptance matrix A as a factor, it is sufficient (but again, not necessary) to show that

$$A_{ji}^{(n+1)} + kA_{ji}^{(n)} = 0.$$

The nth derivative of A can be described element-wise by

$$A_{ji}^{(n)} = F_{ji}^{n} e^{\beta F_{ji}}$$

where the matrix F is given by

$$F_{ji} = \min(0, c_i - c_j);$$

we now have

$$\begin{aligned} A_{ji}^{(n+1)} + kA_{ji}^{(n)} &= F_{ji}^{n+1} e^{\beta F_{ji}} + kF_{ji}^{n} e^{\beta F_{ji}} \\ &= F_{ji}^{n} e^{\beta F_{ji}} \left(F_{ji} + k\right). \end{aligned}$$

For Max-SAT problems (and in fact also for the binary perceptron and spin glass problems), the state costs are integer-valued; this restricts the elements of F to the non-positive integers. If we set k to one (so as to match our observation for the constant schedule), the right hand side is zero when F_{ji} is 0 or -1. Further, we note that for strictly positive β, the RHS shrinks exponentially as F_{ji} decreases; thus the elements for which the rule does not hold (i.e. where $F_{ji} < -1$) contribute less to the gradient calculation. This suggests that the relationship between the derivatives may be caused of the dominance of those transitions with smallest cost difference.

5.1.3 Computational Limitations

The time taken to produce an optimal schedule depends chiefly on three variables:

- N: The number of states in the model; this is usually the number of nodes in the barrier tree for the problem.
- T: The length of the schedule. For simplicity, we shall assume at this point that no coarse-graining is used.
- R: The number of iterations of the optimiser (usually SCG) required for satisfactory convergence.

Each transition matrix consists of N^2 elements; with the exception of the leading diagonal, these may all be computed in constant time. The leading diagonal consists of N elements, each of which requires a sum of N elements to compute (the probability of a self transition includes those of the rejected non-self transitions), and thus each transition matrix may be computed in $O\left(N^2\right)$ operations, and the complete set of matrices requires $O\left(N^2T\right)$ operations. (The derivatives of these matrices are the same order of complexity to compute.)

Having produced a set of transition matrices, the average WYA cost is computed by applying each of the transition matrices to an initial state vector. This is a series of T multiplications between $N \times N$ matrices and a vector, and thus requires $O\left(N^2T\right)$ multiplications. Each of the T components of the gradient is computed via a cost evaluation with one transition matrix replaced by a derivative, thus the gradient is even more costly, at $O\left(N^2T^2\right)$ operations.

BSF cost effectively requires these steps to be repeated with sets of modified transition matrices. The modification itself is not difficult, but the extra repetition increases the computation time. The number of repetitions is equal to the number of distinct costs assigned to the states in the model. In practise, this seems to scale roughly with N, so the BSF cost and gradient require $O\left(N^3T\right)$ and $O\left(N^3T^2\right)$ operations respectively.

Finally, the optimisation process requires R iterations, the time for each of which is dominated by that of the gradient calculations. However, R is an unknown quantity, being largely influenced by the structure of the cost landscape, the choice of search heuristic and cost function, the peculiarities of whichever algorithm is used to optimise the schedule, and the choice of termination criterion. Observations suggest that the changes which increase the computation required for each iteration (e.g. BSF instead of WYA cost, larger problems, longer schedules) tend also to increase the number of iterations required.

Our implementation limits the number of iterations to 500. For larger problems, this limit was frequently exceeded, but it appears that the convergence in those cases had slowed to the point where further iterations produced schedules which were almost indistinguishable from the 500th iteration.

5.1.4 Schedule length

For most problems, schedules of length 200 and longer were used. When a range of schedule lengths is examined (e.g. in §5.4.4), those below a certain length appear to be qualitatively different, typically with such a low temperature that the resulting behaviour is effectively a descent, with the ability to escape local minima sacrificed in return for improved results in the cases where escape is not necessary. Examples of this may be seen in figs. 5.17, 5.18 and in §C.3; in each set of schedules, the shortest temperature schedule (length 100) does not have the same shape as the others, being approximately linear, but having similar starting and finishing temperatures to the longer schedules. The temperatures of both schedules never exceed 0.08; given a temperature this low, the probability of accepting an uphill transition of cost difference 1 (all Max-SAT problems have integer costs) is given by $e^{-1/0.08} \approx 3.727 \times 10^{-6}$, a probability small enough to ignore in the context of a schedule of length 100.

Experiments with the average first passage time show that the average time taken to reach the optimum may be around 100 to 150 iterations (§7.6) on our problems (excluding the hurdle problem). Given that this is average time before the minimum is first reached, it seems sensible that schedule lengths of approximately the same order of magnitude should be of greatest interest. Generally speaking, WYA schedules on 20-variable problems reverted to descent when number of iterations fell below 120 or so. BSF schedules tended to maintain the same form until the number of iterations became trivially small.

5.1.5 Coarse-Grained Schedules

Without coarse-graining, a schedule of length T requires an optimisation over a set of T variables. Because the effect of perturbing individual variables becomes very small, and may be compensated for almost entirely by adjusting "adjacent" variables in the schedule, the optimisation tends to become very slow, and the schedule produced appears to be locally noisy. Unsurprisingly, sufficient coarse-graining avoided this problem, producing a schedule very similar to an averaged version of the noisy original.

Additionally, coarse-graining reduces the computational effort required for each cost evaluation the number of variables involved, thus decreasing the number of iterations (i.e. cost evaluations) required for convergence. Due to the improvements to execution time, most of the results in the section show coarse-grained schedules. Comparisons between non-coarse-grained schedules and coarse-grained schedules of various lengths can be seen in figs. 5.4 and 5.5; observe that in both cases, the coarse-grained schedules are good approximations to the originals.

The most significant problem introduced by coarse-graining is that any features which persist for only a few iterations are likely to be distorted or omitted altogether from a coarse-grained schedule. Such features are not thought to be common, with the exception of the rapid end-of-schedule cooling (§5.4.3). Additionally, it is difficult to verify that longer coarse-grained schedules are good approximations in all situations, especially in the case of BSF schedules, where the computational effort associated with optimisation is considerably higher than for WYA cost.

Regardless of the degree of accuracy with which coarse-grained schedules approximate the originals, it should be noted that the coarse-graining is simply an alternative way of parameterising the annealing schedule, the object being to reduce the number of degrees of freedom. It is not necessary to consider coarse-grained schedules solely as a means of approximation; annealing with a coarse-grained schedule is a viable heuristic in its own right. Other parameterisations are of course possible; we discussed this briefly in §3.3.

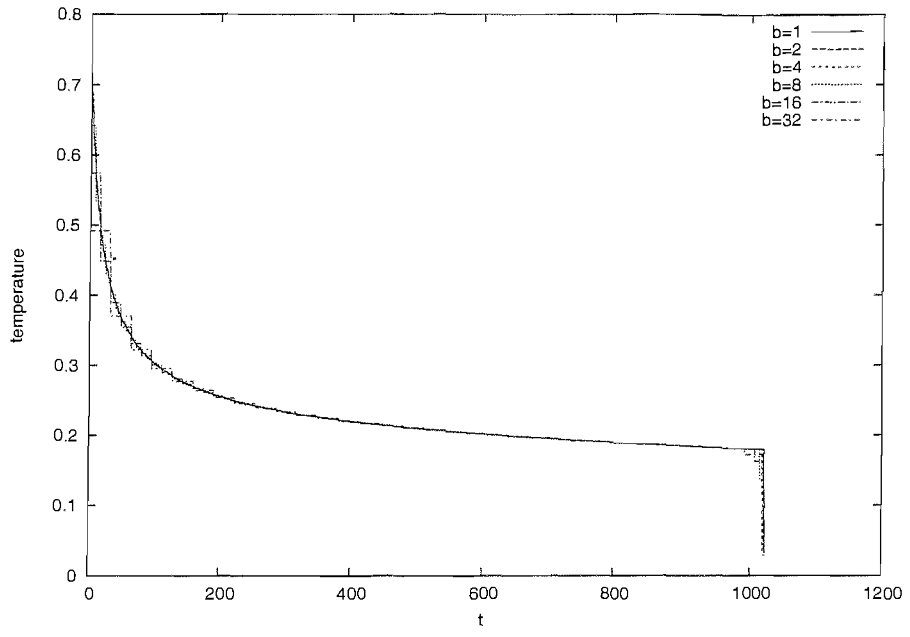

FIGURE 5.4: Coarse-grained WYA schedules for a hurdle problem, various block sizes.

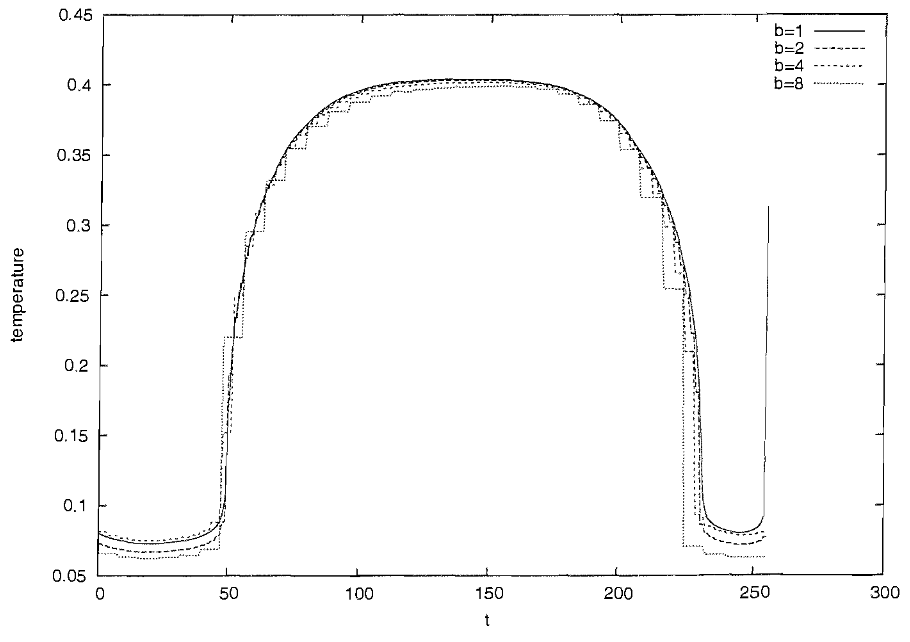

FIGURE 5.5: Coarse-grained BSF schedules for a Max-SAT problem, various block sizes.

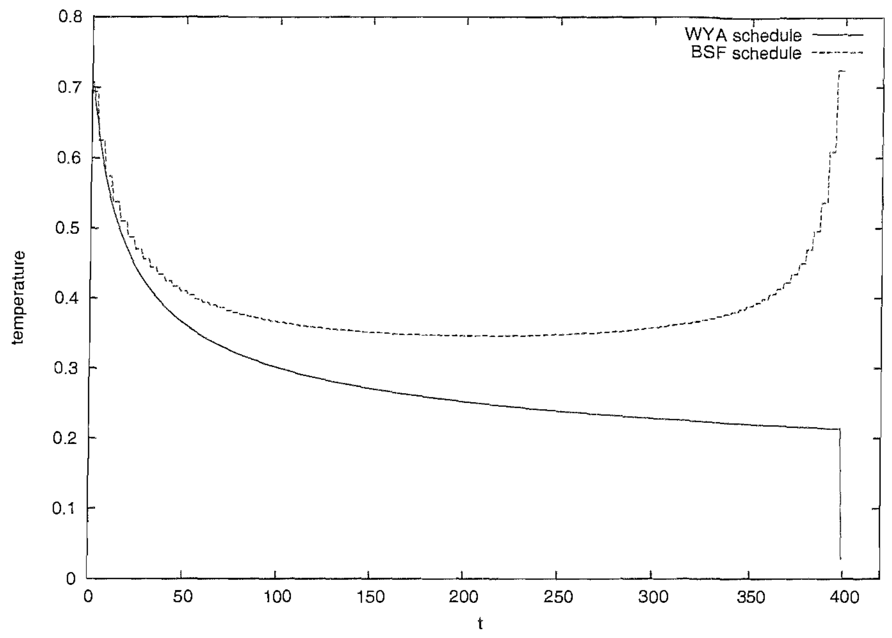

FIGURE 5.6: Example WYA and BSF-optimised schedules for a 16-bit hurdle problem

5.2 WYA and BSF Schedules for the Hurdle Problem

Schedules were produced for a 16-bit hurdle problem, optimised separately for WYA and BSF cost (fig. 5.6). Note that there is little similarity between the schedules except for during the first 20 or so iterations, after which the WYA schedule continues to cool, but the BSF schedule eventually begins to increase temperature, rising to a peak at the end of the schedule. While we know that the significance of the final iterations of a BSF schedule decreases towards zero, possibly making the end of the schedule very noisy, we would not have expected the temperature to increase.

In contrast to WYA cost, convergence of a BSF search is counter-productive; for the search to spend a large number of iterations in or around a local minimum is of no additional value. The strategy which appears to be favoured in our BSF hurdle schedules is a best-effort attempt to get close to the global minimum (using about half of the available time), followed by a local search. The rapidly increasing temperature towards the end is a way of maximising exploration of the surrounding region, at the cost of potentially straying away from it.

These two optimised schedules were tested on the hurdle problem by running a simulated annealing algorithm with the suggested temperature schedules. For each schedule, the annealing process was repeated 100 000 times, with each run beginning at a point chosen randomly from the search space. The average WYA and BSF costs were computed for both schedules, and are shown in figs. 5.7 and 5.8.

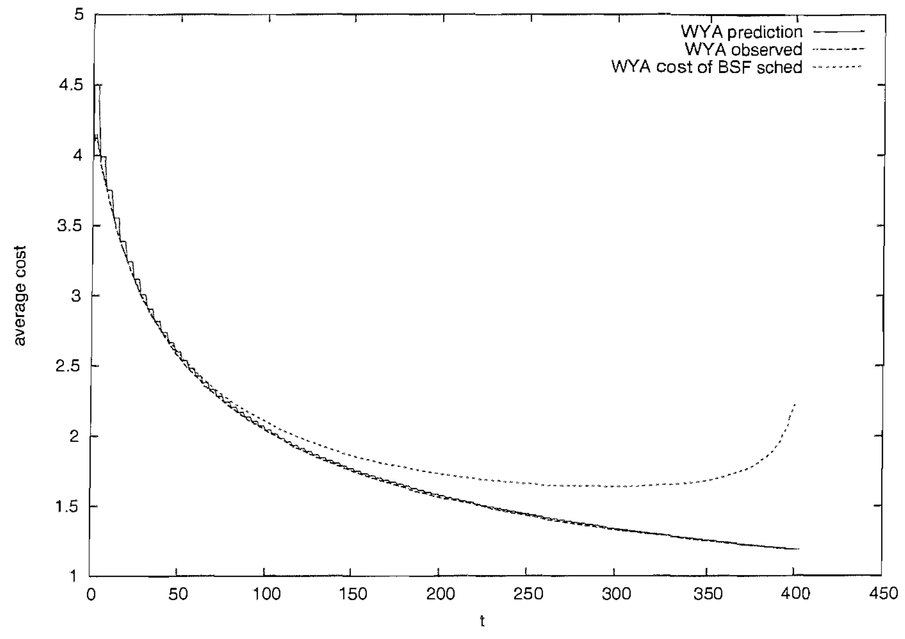

FIGURE 5.7: Predicted vs. measured performance of a WYA schedule; measured results averaged over 100 000 runs.

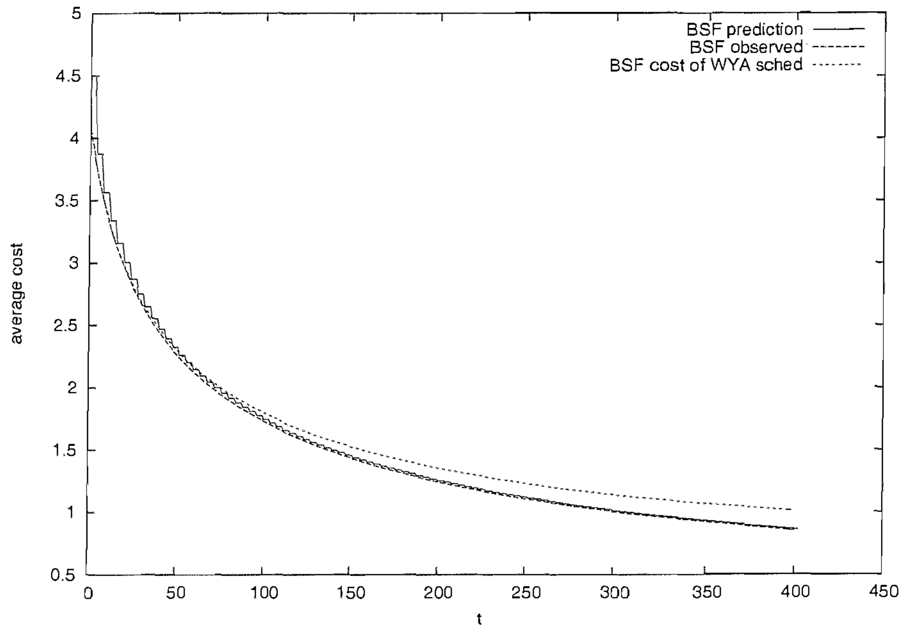

FIGURE 5.8: Predicted vs. measured performance of a BSF schedule; measured results averaged over 100 000 runs.

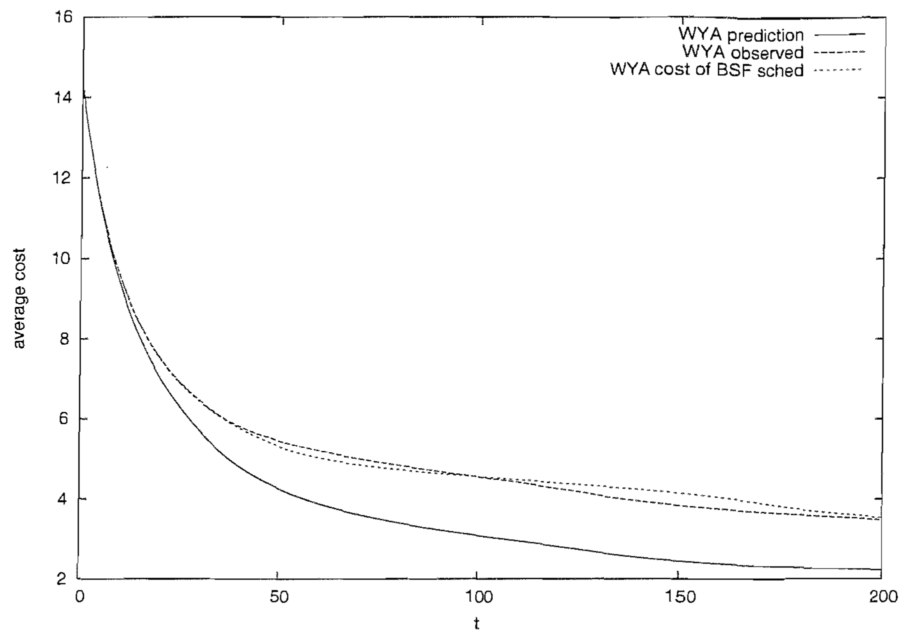

FIGURE 5.9: Predicted vs observed performance of a WYA-optimal schedule for a SAT problem

Most notably, we see that the average costs observed from the simulation exactly match those predicted by the model, at least up to the effects of coarse-graining. This is however not so surprising, as the reduced model for the hurdle problem is an exact model of the original problem (§4.1).

For comparison, we show also the WYA evaluation of the BSF schedule, and vice versa. The WYA schedule does not give unreasonable BSF performance, but the converse is not so. Due to the increasing temperature towards the end of the BSF schedule, the WYA performance produced is rather poor.

5.3 Models of Real Problems

Aside from the hurdle problem, optimised schedules were also produced for state-amalgamated models of instances of the Max-SAT, spin glass and binary perceptron problems, using barrier tree data produced by Jonathan Hallam. Various sizes of problem were used, mostly with search spaces of between 2^{15} and 2^{20} points, represented by barrier tree models. On all of these problems (results summarised in §C.2.5), it was found that the predicted costs were under-estimates; the optimised schedules did not perform so well in practise as the predictions of the model suggested. Figures 5.9 and 5.10 compare the predicted and actual cost of WYA- and BSF-optimised schedules (themselves shown in fig. 5.11) for a 20-variable Max-SAT problem.

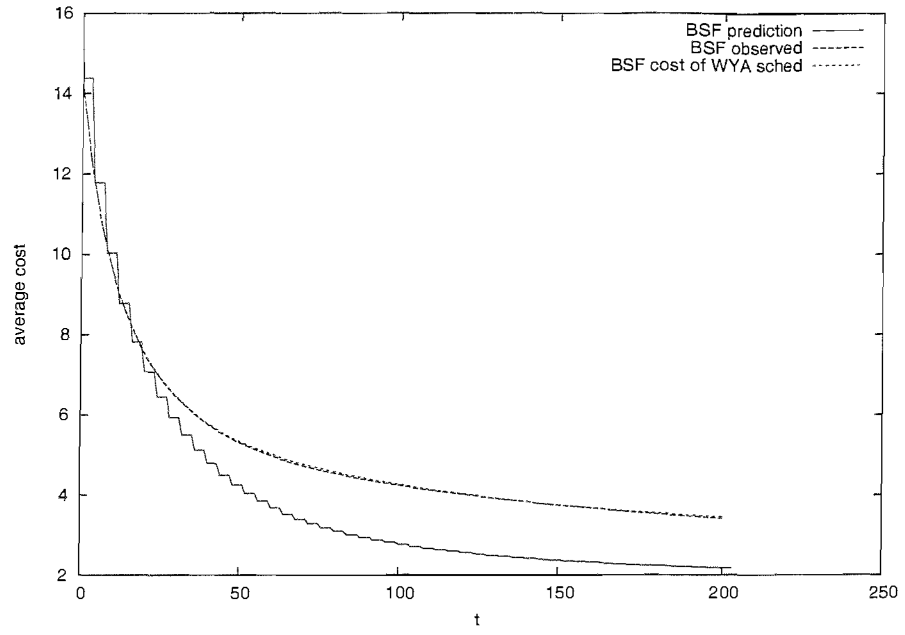

FIGURE 5.10: Predicted vs observed performance of a BSF-optimal schedule for a SAT problem

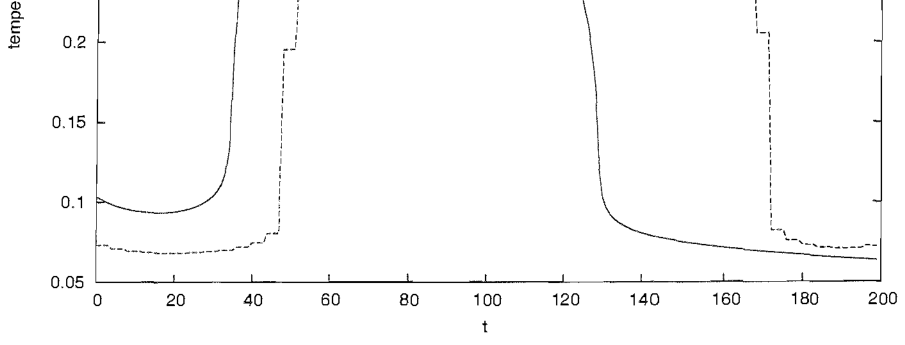

FIGURE 5.11: WYA and BSF-optimal schedules for a 20 variable Max-SAT problem

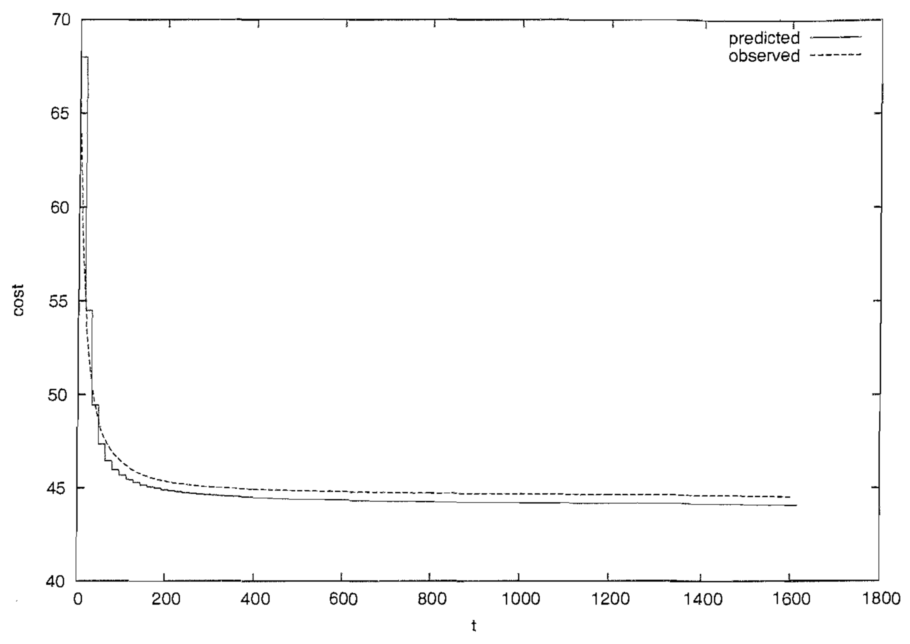

FIGURE 5.12: Predicted vs observed performance of a WYA-optimal schedule for a spin-glass problem

The fact that the observed results differ from those predicted by the model is no real surprise, as the barrier tree representation is an approximation to the actual problem (§4.3.2). However, the fact that the predicted performance is *always* over-estimated is worthy of investigation, as it suggests the presence of a systematic effect which causes the model problems to become easier to solve than the problems themselves. This under-estimation of cost occurred also for the binary perceptron and spin-glass problems (fig. 5.12), and is therefore unlikely to be peculiar to our chosen set of problems.

Aside from the approximation inherent in using a barrier tree (instead of the actual search space), all other aspects of the model are "exact", capturing accurately every detail of the heuristic's behaviour; this assertion is supported by the accuracy of the model in predicting the costs for the hurdle problem (§5.2). It therefore seems logical to conclude that the error we see is a product of the barrier tree approximation. Substituting the digraph of level-connected sets in place of the tree of level-accessible sets provides a similar result (fig. 5.13), suggesting that the harmful approximation lies at least partly in the grouping of search points into level-connected sets, rather than the further amalgamation of those sets into barrier tree states.

Although most of the test problems used were based on 20-bit binary string representations, for which the barrier tree model was constructed through exhaustive mapping of the cost landscape, it is possible to produce approximate models through random sampling of the search space. The schedule shown in fig. 5.14 was produced in this way, but has a very similar structure to those produced for the smaller problems.

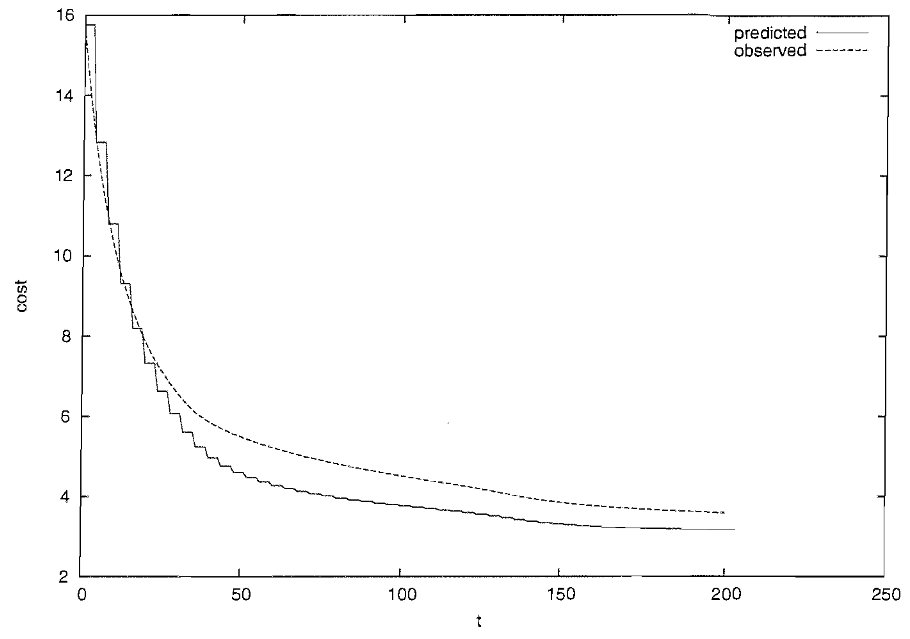

FIGURE 5.13: Predicted and actual costs for an annealing schedule optimised (for WYA cost) over the digraph of level-connected sets (instead of the barrier tree) for a 20-variable Max-SAT problem

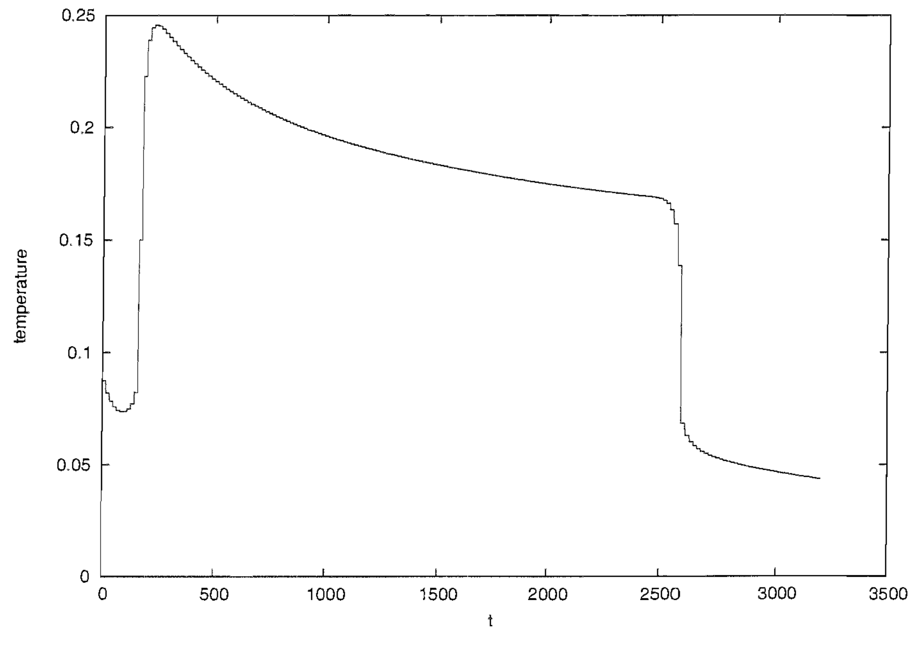

FIGURE 5.14: An annealing schedule optimised for WYA cost a 40-variable Max-SAT problem, produced from a sampled barrier tree model with 85 states.

5.4 Features Observed in Optimised Annealing Schedules for Max-SAT Problems

The optimised schedules vary considerably depending on the cost function, the search heuristic, and the type of problem. The main points of variation appear to be the behaviour in relatively small regions at the beginning and end of the schedule, the rest of which (with a few exceptions) usually resembles either some form of decay curve (WYA schedules), or a convex curve, occasionally oscillating slightly (BSF schedules). The WYA and BSF Max-SAT schedules shown already (figs. 5.5, 5.11 and 5.14) show typical behaviour. Optimal WYA and BSF schedules for all of the Max-SAT problems are shown in §C.2.2; in most cases, the Max-SAT schedules for the same cost function have a similar form. The exception is discussed in §5.5.

5.4.1 Cold Start

For the Max-SAT schedules, an initial period of low temperature was common, effectively reducing the start of the search to a descent; this was observed for both WYA and BSF schedules. Since this behaviour does not appear to occur with the hurdle problem it is likely that this effect is caused by the structure of the cost landscapes for the Max-SAT problems. It is likely (especially given the observations in §4.4) that the local minima are concentrated around the global minimum, causing large regions of the landscape to be easily traversable by descent. The length of this "cold start" is likely to represent the number of iterations before uphill steps become useful, i.e. the average number of iterations of descent required to move from a randomly chosen point to a local minimum. We do not see this behaviour on the hurdle problem, probably due to the very uniform distribution of minima. On average, half an iteration is sufficient to move from a randomly chosen point to a minimum of the hurdle problem.

This feature was also observed in optimal schedules for graph partitioning problems by Strenski and Kirkpatrick (1991), who provide an insightful alternative explanation in terms of the temperature of the initial distribution. The gradual cooling seen in many annealing schedules gives the system time to equilibrate at each temperature; however, this cannot begin immediately when the initial distribution is effectively at the infinite-temperature limit. To remedy this, the optimal schedule dedicates the first few iterations to bringing the distribution reasonably close to an equilibrium state, and the fastest way to do this is by setting the temperature to zero.

5.4.2 Gradual Cooling

Excluding short sections at the beginning and end, all of the WYA annealing schedules provided an asymptotically decreasing temperature; taken in isolation, this feature matches "traditional" annealing schedules, which use a strictly decreasing temperature. The usual justification is that if the temperature is reduced slowly enough, then the global minimum can be reached with certainty arbitrarily close to one (Hajek, 1988).

5.4.3 End-of-Search Behaviour

When optimising a schedule for WYA cost, the goal is to minimise the cost of the last state reached. This usually coincides with the goal of approaching the global minimum, however the two interests diverge at the end of the schedule. When the number of remaining iterations becomes very small, minimising the cost of the current state becomes much more important than exploring the landscape, and thus the annealing temperature drops rapidly.

The extreme case is the final iteration of the algorithm. An uphill transition on the final iteration of a WYA schedule should never occur, as this would reduce the final solution quality; it is therefore no surprise that the final-iteration temperature is roughly[1] zero. For a BSF annealing schedule, the last temperature in the schedule has no effect at all on the search; the point tested on the last iteration will affect the solution quality only if it has cost lower than that of the penultimate point, in which case it is guaranteed to be accepted, regardless of temperature. Thus the last temperature value in a BSF schedule is entirely irrelevant, and should be disregarded.

In the case of the hurdle problem, the final period of rapid cooling is limited to a single iteration. This is because the bipartite structure of the cost landscape (§4.1) is such that every point in the search space is either a minimum, or has a neighbour set composed entirely of minima. A single iteration at zero temperature is therefore sufficient to ensure that the search has terminated at a minimum. All of these effects are of course masked somewhat when coarse-grained schedules are used.

5.4.4 Overlapping schedules

So far, we have compared optimised schedules in various scenarios, discussing both features of the schedules and their actual performance. One aspect not yet covered is the relationship between schedules of different lengths, optimised for the same problem. The behaviour in this regard was found to be largely problem-specific.

The most surprising result was the similarity between WYA hurdle problem schedules, with schedules of differing lengths "overlapping" almost perfectly (fig. 5.15). With the exception of the final iteration of each schedule, for which the usual drop in temperature

[1]In our experiments, this temperature was slightly nonzero. When temperature is sufficiently small that the acceptance probability for any possible uphill transition is negligible, the effect of further modifying that temperature is greatly diminished. This effect, combined with the decision to optimise *inverse* (i.e. reciprocal) temperature, prevents the SCG optimiser from actually reaching zero.

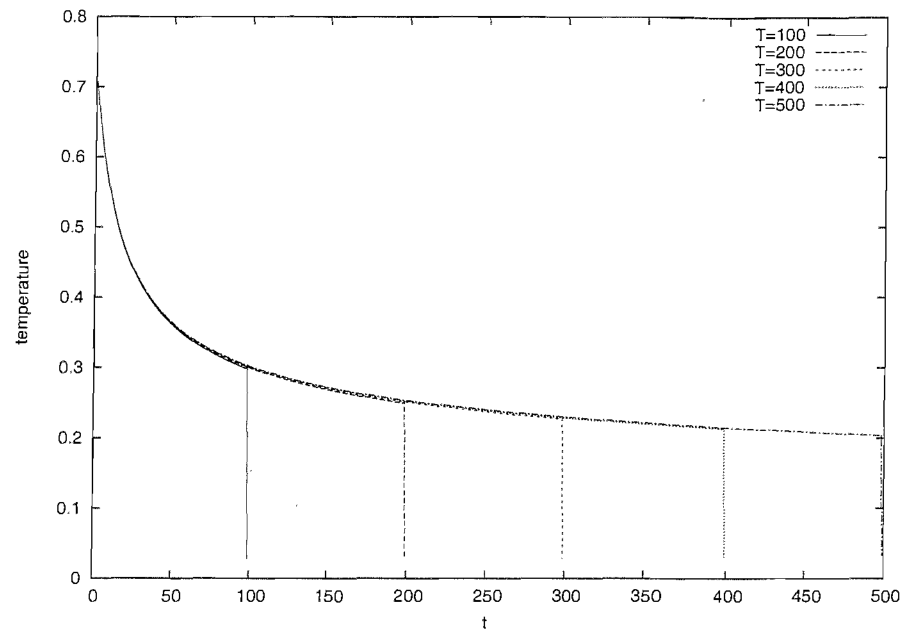

FIGURE 5.15: WYA hurdle schedules show a very good overlap

occurred (§5.4.3), the schedules match so well that they appear almost coincident. We note that a similar form of overlap was seen in Hoffmann and Salamon (1990), but with the ends of the schedules (instead of the beginning) sharing the common behaviour. Schedules of various lengths have also been seen to rescale onto the same curve in Christoph and Hoffmann (1993).

This is of particular interest; if a short, optimised schedule may be used in some way to extrapolate a longer schedule, or indeed, schedules of arbitrary length, then not only do we have a computational shortcut, but the nature of the relationship between the schedules may be correlated with properties of the landscape, potentially allowing schedules to be produced for other problems based only on measurements of those properties.

Unfortunately, this phenomenon appears to be confined to WYA hurdle schedules, and is thus more likely to be connected with the self-similar[2] structure of the hurdle landscape, rather than an inherent dependency between schedules of differing lengths.

An equivalent comparison for BSF hurdle schedules (fig. 5.16) does not show the same effect. Although the BSF schedules are very similar to each other, they are clearly distinct, departing within the first 50 iterations. Despite this, they each seem to follow the same general 'U' shape, and it is quite possible that they may be scaled onto each other in some way, or approximated by a general formula.

[2]At each minimum, the size of the uphill step (i.e. the cost difference) required to move towards the global minimum is identical, only the number of possible "correct" neighbours changes. At each non-minimum point, all neighbours are of lower cost, so Simulated Annealing effectively chooses one at random.

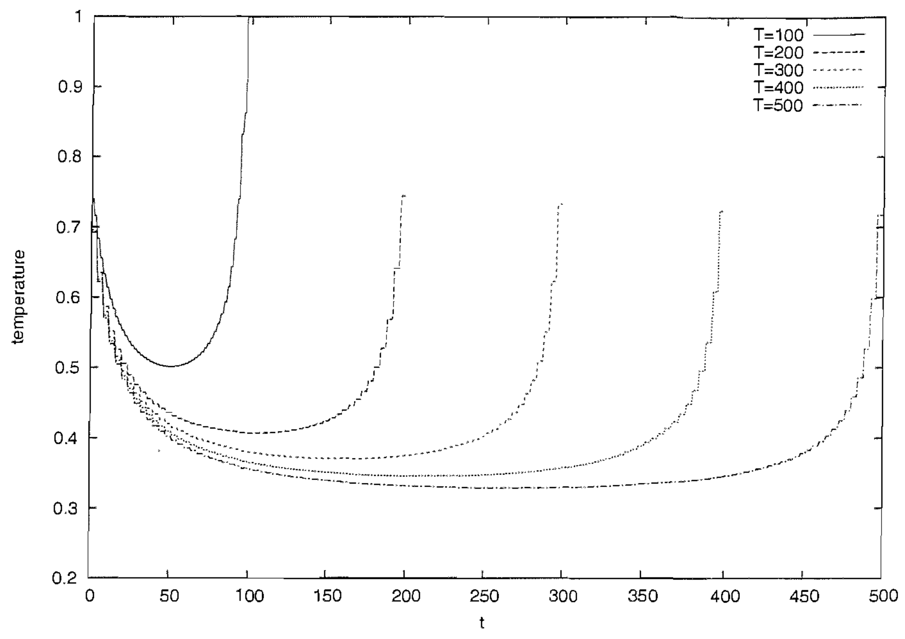

FIGURE 5.16: BSF hurdle schedules of differing lengths are of similar shape, but do not overlap

For certain problems, BSF schedules displayed a slight oscillatory behaviour (fig. 5.17). It is possible that the periodic behaviour effectively causes restarts in the search, reproducing the effect of an iterated descent algorithm. The small amplitude of the fluctuation suggests that a full restart is not being performed, but rather a backtrack to a higher barrier state. Periodically spiking schedules have been seen before in Boese and Kahng (1994), while Bölte and Thonemann (1996) produced optimised schedules which included a cosine component.

Finally, fig. 5.18 shows a series of Max-SAT schedules. Here we observe that the low-temperature regions towards the end of the schedules are very similar; despite being of different lengths, these regions seem to have approximately the same slope. Although the initial temperatures differ somewhat, all of the schedules appear to peak at the same time ($t \approx 60$), adding some weight to our assertion (§5.4.1) that the initial low-temperature period represents a problem-specific period of descent, after which the landscape becomes more difficult to navigate. The peaks are of differing height however, and while the graphs do appear to have the same gradient shortly after, the cooling process eventually slows down, giving the longer schedules a concave shape. Thus despite not overlapping, the schedules do share some similar properties, and longer optimal schedules are likely to follow a predictable form.

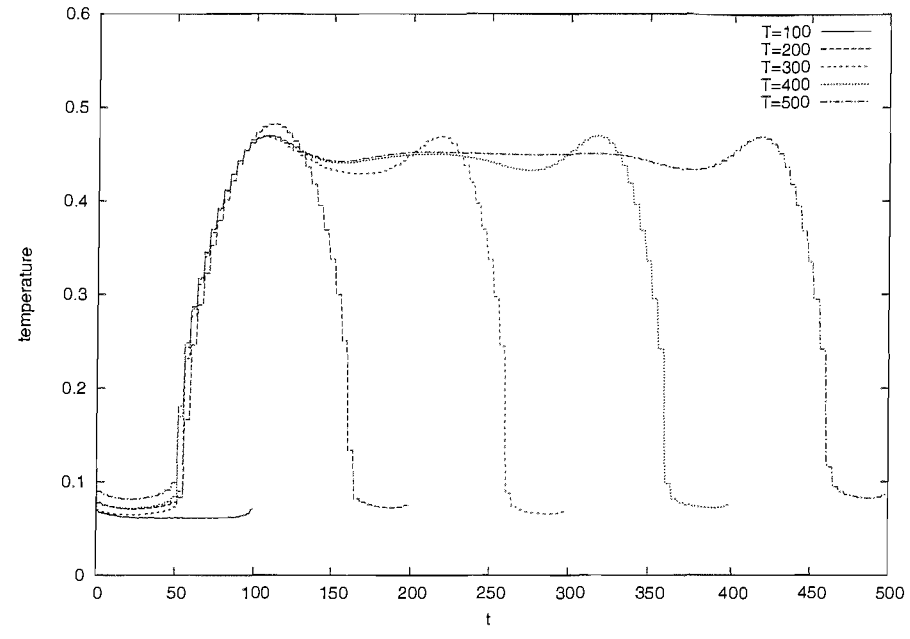

FIGURE 5.17: BSF schedules on a Max-SAT problem were sometimes found to have a slightly periodic behaviour. Note that the amplitude of the oscillation is greatest for smaller schedules, and almost vanishing for the longest.

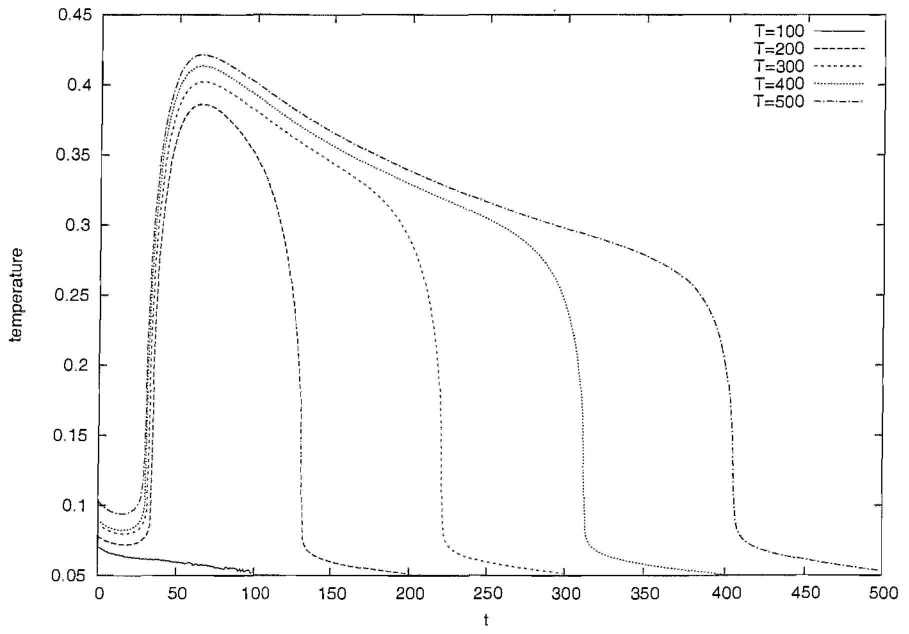

FIGURE 5.18: WYA schedules of various lengths for a 20-variable Max-SAT problem.

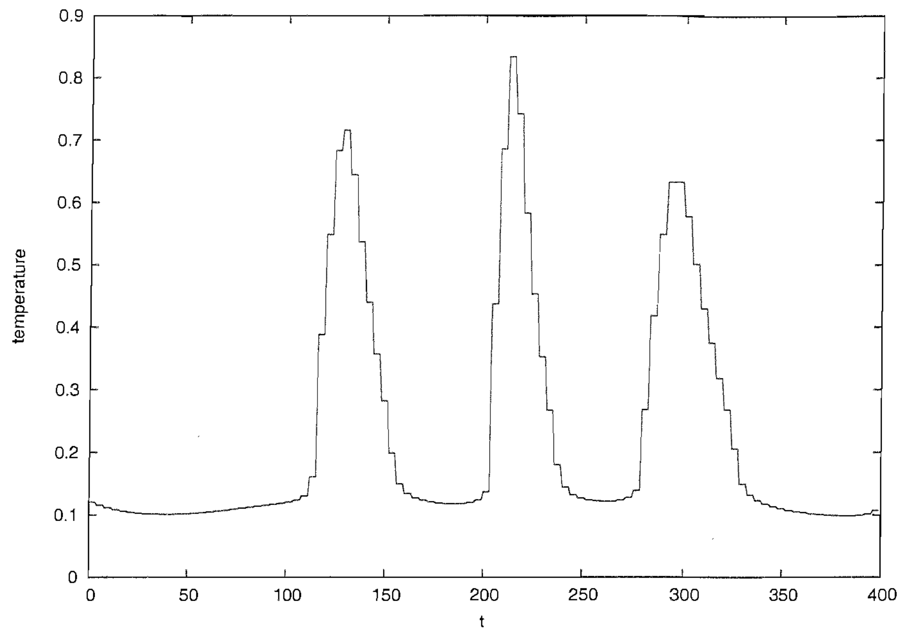

FIGURE 5.19: A Max-SAT BSF schedule with three similarly-shaped peaks.

5.5 Periodic Schedules

Several of the BSF-optimised schedules showed particularly interesting periodic behaviour. Over the Max-SAT problems, this was limited to a single instance (fig. 5.19), in which there are three distinct temperature spikes. The three peaks have similar (but not identical) height and width; in each case the increase is visibly steeper than the subsequent decrease. Except for the three peaks, the temperature is very low.

The problem on which this schedule was optimised (number 18, fig. C.1) is unremarkable except for the relatively small number of minimum states; in fact it has fewer than any of our other Max-SAT instances. This suggests the possibly explanation that the problem is easy enough that descent performs very well. The first 100 iterations of descent should provide enough time to for at least one visit to the global minimum, provided the search has not become "snagged" on any local minima.

As the probability of visiting the global minimum increases, the expected benefit of performing further iterations of descent diminishes. Since there is a non-zero chance that the descent may have become trapped at a local optimum, there will come a point (at $t = 110$ in this case) where attempting to free the "trapped" probability mass (by increasing the temperature) offers a better return on invested iterations than continued descent. Of course, not all of the probability mass is freed (i.e. there is a nonzero chance that an annealer was stuck in the minimum, and fails to permanently escape it even with the aid of the temperature increase); thus the process is repeated until insufficient time remains for another complete cycle.

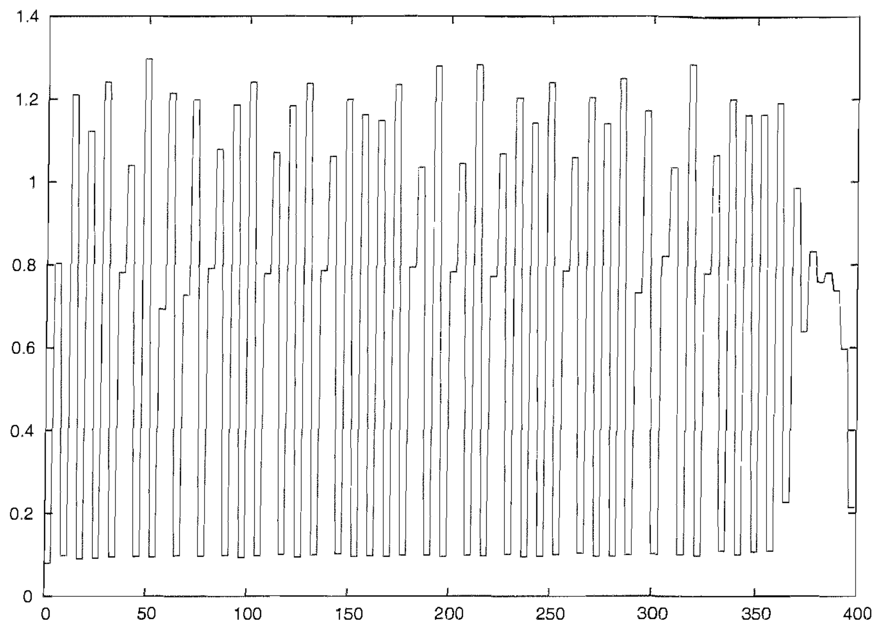

FIGURE 5.20: A BSF-optimised binary perceptron schedule (problem '10-12-2'), alternating between high and low temperatures.

Periodic behaviour was also seen in several binary perceptron schedules. The most extreme example is shown in fig. 5.20. On first inspection, the schedule may look like random noise; note however that the temperature values which appear are very well clustered into three groups. There is a set of near-identical "low" values around 0.1, a set of "medium" values close to 0.8, and a more varied "high" range between 1 and 1.3. For the vast majority of the schedule, there are no two adjacent temperatures from each group (up to the effects of coarse-graining of course). Further, the "medium" temperatures only occur occasionally (ignoring the endpoints of the schedule) as intermediates between a low-high transition.

This supports our earlier speculation (§4.4.2) that the binary perceptron problems contain a large number of shallow[3] minima; under this assumption, the best strategy could well be to oscillate between descent and random walk. This schedule was produced with a coarse-graining block size of 4. Ignoring for a moment the occasional intermediate temperatures, the schedule effectively performs four iterations of descent, followed by four steps with temperature around 1.2, giving around a 40% chance of accepting an uphill step of size 1. This could give a reasonable chance of escaping the minimum (depending on the various costs of the neighbour states), after which the descent would provide a chance to progress lower.

[3]The "depth" of a minimum could be described as the minimum increase in cost required to escape the basin of attraction of that minimum. In this discussion, we take "shallow" to mean "has only a trivial basin of attraction", with all neighbours being non-minima.

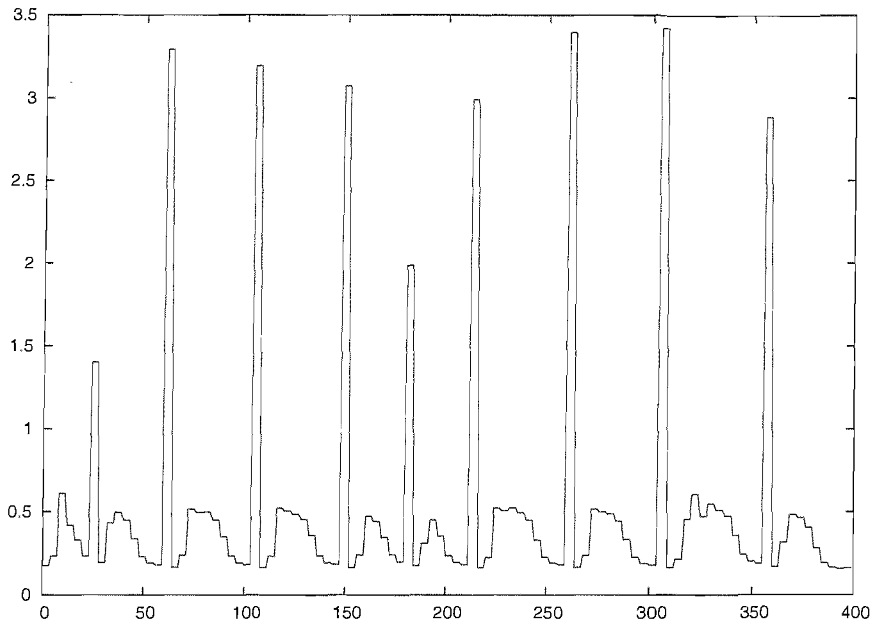

FIGURE 5.21: A BSF-optimised schedule for binary perceptron problem '10-12-1'. A series of high-temperature spikes is interleaved with a series of lower-temperature humps.

A further interesting example is given by fig. 5.21, in which the schedule contains a repeated pattern of approximately 50 iterations in length. This pattern contains four iterations at very high temperature (four is the coarse-graining block size, and hence the minimum width for such a spike), followed by a much smaller "hump". This hump varies in shape (this is likely due to noise in the SCG optimisation), but in almost every case begins with a sharp increase, then cools slowly, then more rapidly, then more slowly again, until the next spike.

It is interesting that the temperature immediately after each spike drops very low, then rises before cooling. This temperature drop is reminiscent of that seen in the non-periodic Max-SAT BSF schedules, as is the maximum temperature of the humps (0.5 compared to 0.4). The similarity might not be coincidental; each temperature spike provides four iterations during which an uphill transition of size 1 is accepted with between 50% and 70% probability. These spikes may well be enough to induce a periodic backtrack of several steps, causing the probability distribution to disperse a little. That the behaviour between spikes is not a descent suggests that avoiding local minima is a non-trivial task; the shape of these subsections may be the optimal schedule for traversing some particular subspace of the problem.

In several cases, WYA-optimised perceptron schedules began with periodic spikes of decreasing magnitude, followed by a gradual cooling phase, similar to that seen in Max-SAT WYA schedules. The two phases to this distribution could correspond to groups of minima at different temperatures; the periodic spikes may be intended to push the search into the deepest branch of the tree at an early stage, while the slower cooling then

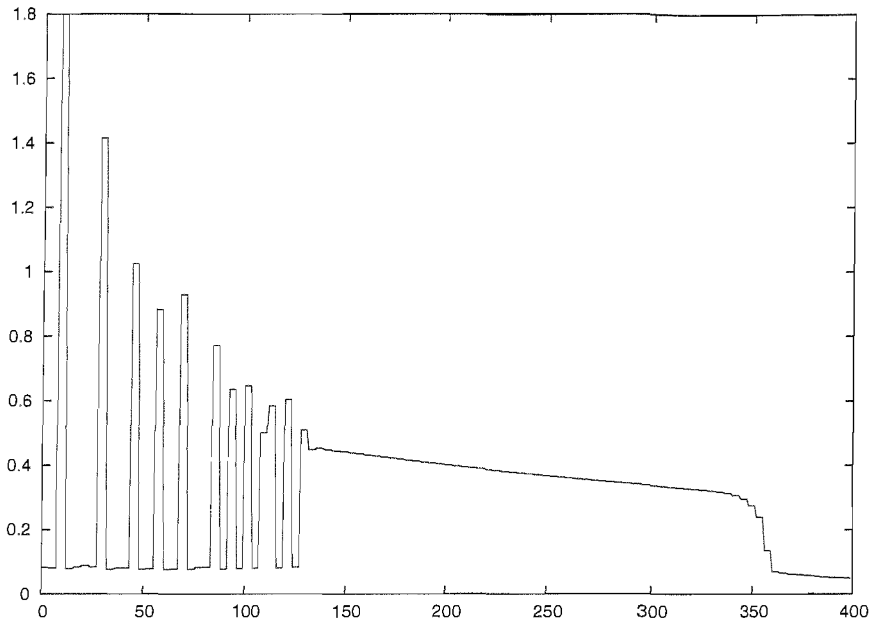

FIGURE 5.22: WYA-optimised schedule for binary perceptron problem '10-12-1'. An oscillating temperature is followed by a gradual cooling.

attempts to ensure that, whichever branch was eventually chosen, the search finishes in the lowest state of that branch.

5.6 Variable Mutation

The variable mutation search was found to produce considerably better performance than simulated annealing on the hurdle problem (fig. 5.23), although this advantage disappeared when the two were compared on Max-SAT problems. This is largely due to the structure of the hurdle problem, which causes transitions towards the global optimum to become increasingly difficult. Variable mutation search benefits from a "ratchet effect", as it will never move to a state of higher cost. On many problems this would make traversal of barriers very difficult, requiring many mutations in one step. However, all local minima on the hurdle problem are very small and shallow, and can be escaped with a double mutation. Simulated annealing, however, is likely to take backward steps when close to the global optimum, simply because they outnumber by far the opportunities for steps towards the optimum; this is in agreement with the observations in Prügel-Bennett (2004b).

Fig. 5.24 shows a large qualitative difference between the mutation schedules for the hurdle and Max-SAT problems. Both begin with a high mutation rate, reflecting the idea that a small period of random search can rapidly provide an improvement over the starting point. Afterwards, the mutation schedules roughly resemble the corresponding annealing schedules.

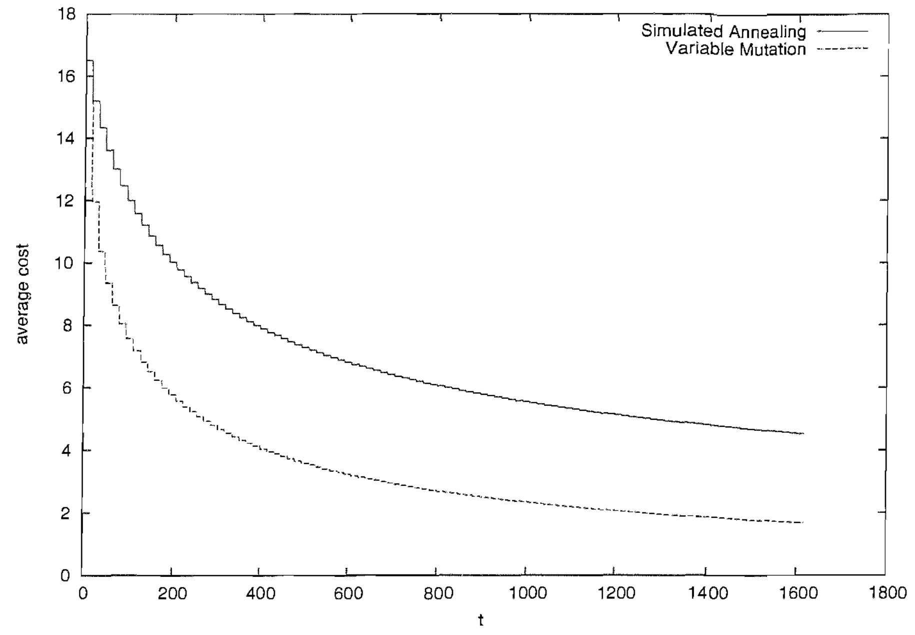

FIGURE 5.23: Comparison between performance of WYA-optimised annealing and mutation schedules, on a 64-bit hurdle problem.

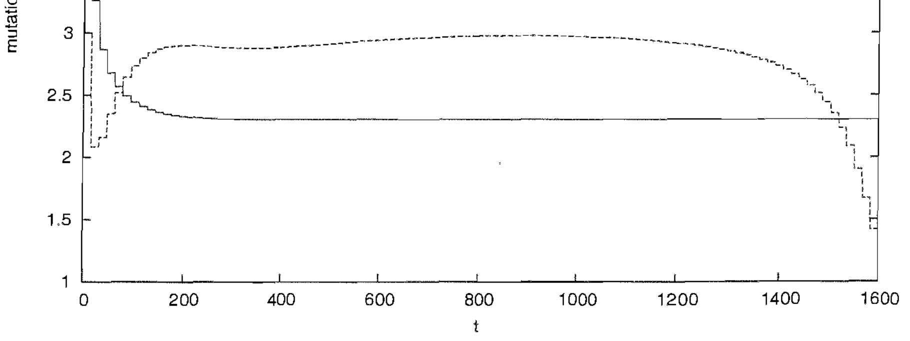

FIGURE 5.24: Example WYA-optimised mutation schedules for two problems.

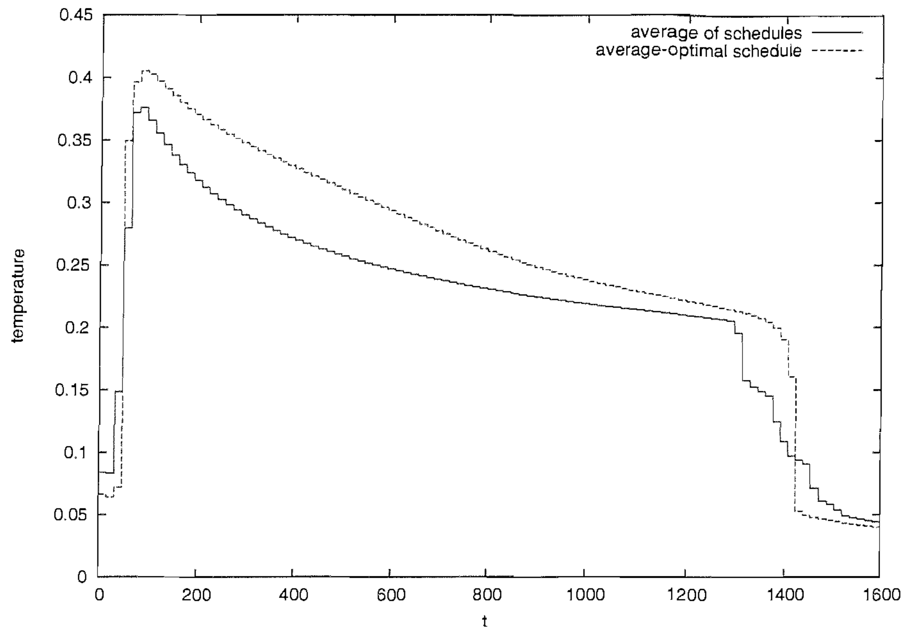

FIGURE 5.25: Comparison between an average-optimal schedule and the average of all the individually-optimal schedules. Schedules produced for WYA cost of simulated annealing on a set of twelve Max-SAT problems; 100 (constant-temperature) blocks of 16 iterations.

5.7 Average-Optimal Schedules

In this section we explore the possibility of optimising the parameters of a heuristic for a class of problems rather than for a specific instance.

Thus far, our method for producing optimal schedules has been dependent on prior knowledge of the structure of the search space. Disregarding toy problems where the structure (and usually the location of the optimum) can be derived directly from the problem specification, the gathering of this information usually requires an exhaustive enumeration of the search space. Optimising a search heuristic for a problem to which one already has a solution is useful for theoretical study of the quality of the model, but is of very little direct practical use.

However, it may be the case that a heuristic optimised for one instance in a class of problems may also perform well on other instances of that class; ideally, one could hope to find a parameter set which is optimal for a class of problems. Although it is very unlikely that a particular set of parameters will be optimal for every instance in a class, it *is* possible to find parameters optimal for a randomly-chosen instance, by minimising the cost when averaged over all possible instances.

Shown in fig. 5.25 is an annealing schedule optimised (for WYA cost) over a set of twelve Max-SAT problems; we shall refer to this as the *average-optimal* schedule. For comparison, the average of the twelve individually optimised schedules is shown; we

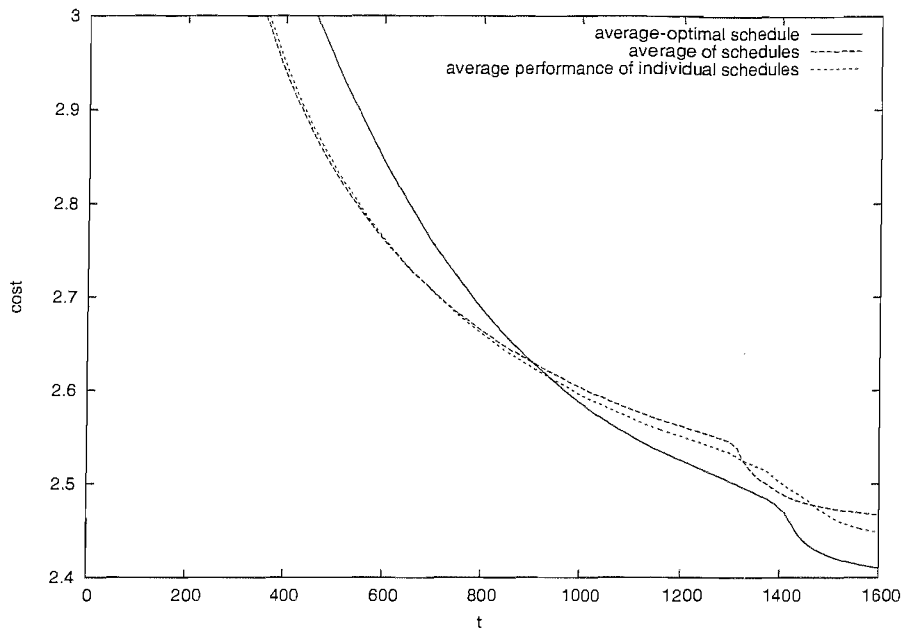

FIGURE 5.26: When tested against the actual problems, the average-optimal schedule gives *better* performance than schedules specifically optimised for those problem instances.

observe that the averaging has produced a schedule which is qualitatively different from the individual schedules, with a visible "step" effect caused by the different times at which the schedules enter the final rapid cooling stage. In contrast, the average-optimal schedule bears much resemblance to the individual schedules.

Since the "stepping" effect is an artifact of the way we have averaged the schedules (directly, by separately averaging each $\beta(t)$), it is possible that the average-optimal schedule may be better approximated by the construction of a schedule with various properties (e.g. time of final cooling, maximum temperature, average rate of decay) chosen to represent the average of those of the individual schedules. Obviously such a construction makes assumptions about the general shape of the schedules, and may be difficult to produce automatically.

Results for the performance of the average-optimal schedule on the real problem were somewhat surprising; although we were aware that the behaviour predicted by the model was subject to a consistent inaccuracy (§5.3) and that the schedules were unlikely to be optimal, it was nonetheless unexpected that the average-optimal schedule should outperform the individual schedules (fig. 5.26) in the majority of cases (ten out of twelve test problems).

Since the differences between predicted and observed performance stem from the loss of information inherent in adopting a reduced-size approximation to the search space, we propose the possible explanation that our optimal schedules are maximising predicted performance by exploiting problem-specific artifacts of the the barrier-tree representation. When minimising the average cost, the benefits of exploiting such an artifact are outweighed by the detrimental effect on the performance on all other problem instances.

5.8 Reduced-parameter optimal schedules without search space modelling

In §5.3, we established that the behaviour of a heuristic on a barrier tree model of a problem differs somewhat from that of the same heuristic on the original problem, with the difference being attributed to the differences between the barrier tree model and the actual search space. Having accepted that the predictions of the model are likely to be inaccurate for this reason, it would still be useful to know whether features such as the "cold start" (§5.4.1) still exist in the "true" optimal schedule (i.e. that schedule which is optimal for the original search space rather than the model).

Of course, finding an optimal schedule on the real search space is too difficult a task; indeed, avoiding this problem was a large part of our motivation for reducing the search space to a barrier tree model. Cost evaluation requires averaging over many runs, with the accuracy increasing only with the square root of the number of runs. Further, the gradient is not directly available, although it may be computed (expensively) through finite difference methods. For these reasons, a schedule with a large number of parameters will (with gradient information being unavailable) take too many steps to optimise, but a very restrictive parameterisation may have a reasonable optimisation time.

We assume that the WYA-optimal schedule for a Max-SAT problem is of the same form as that produced by our model, having an initial low temperature, then a sudden spike followed by gradual cooling, then a final low temperature period. Specifically, we assume that the temperature schedule is of the form

$$\beta(t)^{-1} = [t \geq t_1]\,[t < t_2]\,|A - B\,(t - t_1)|$$

where A is the maximum temperature and B is the rate of (linear) decay. The times t_1 and t_2 are the points at which the initial temperature spike and the final cooling occur; outside the interval $[t_1, t_2)$, the temperature is zero[4]. The modulus operation ensures the temperature is always non-negative, without rendering t_2 ineffectual (as would sometimes be the case if we simply replaced negative temperatures with zero).

The four parameters were optimised for a Max-SAT problem with a schedule length of 400, using the Nelder-Mead Simplex Algorithm. The optimisation was found to be very sensitive to sampling noise, with millions of runs needed before the average was

[4]In fact, we use a slightly positive temperature, to avoid division by zero.

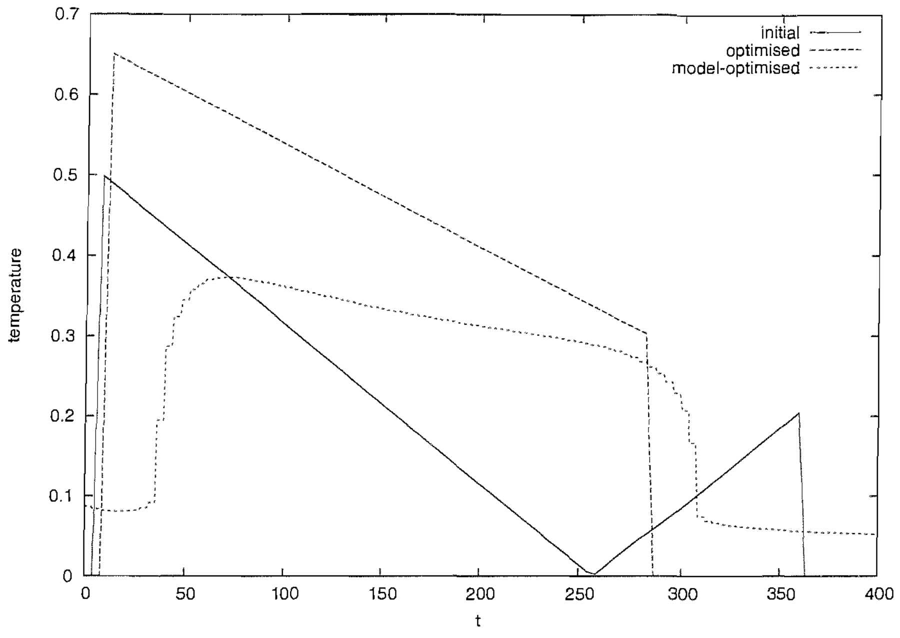

FIGURE 5.27: Comparison between a schedule freely optimised on a barrier tree model, and a four-parameter schedule optimised directly; schedule length is 400. Also shown is the starting point for the latter optimisation, with $t_1 = 20$, $t_2 = 360$, $A = 0.5$ and $B = 0.002$.

	t_1	t_2	A	B	Real cost	Model cost
Initial point	7.326	363.6	0.500	0.00200	2.9424	2.1092
Real-optimal	9.018	284.6	0.654	0.00128	2.7842	2.1759
Model-optimal (est.)	40.00	305.0	0.385	0.00045	3.7010	2.0788

TABLE 5.1: Costs and parameters for schedules optimised on the model and directly. The (arbitrarily chosen) starting point for the latter optimisation is also listed for comparison.

sufficiently accurate. In response to this, an "exact" simulation was implemented, where the probability distribution over all 2^{20} states was transformed at each iteration.

The resulting directly optimised schedule (fig. 5.27) bears some resemblance to the barrier-tree-optimised schedule, although much of this is of course induced by the choice of parameterisation. The "cold start" is still present, but to a much smaller degree, suggesting that while the beneficial effect may have been exaggerated, it does indeed generate a small improvement on the real problem, or at the very least does the search no harm. Similarly, while the cooling period at the end of the schedule does not begin at the same time as for the model-optimised schedule, the difference is only around 20% the length of the cooling period.

The costs of the schedules (table 5.1) show again that the model usually under-estimates cost. The apparent lack of correlation between predicted and obtained cost is disappointing, with the initial point for the search (chosen somewhat arbitrarily) receiving a superior cost from the model than was given to the directly optimised schedule. Similarly, the real cost of the model-optimised schedule was highest of the three, by quite some margin.

We note that the temperature for the directly-optimised schedule is generally much higher than that of the model-optimal schedule. That the model is suggesting temperatures which are too low is an indication that the mobility of the search is being over-estimated, or equivalently that the difficulty of escaping minima and plateaus is being under-estimated. This is in agreement with the consistent under-estimation of cost (§5.3, also observed by Hallam (2006) when modelling descent), and is discussed further in §8.1.

Chapter 6

Parallel Simulated Annealing

In this section we will describe a population-based variant of simulated annealing, "Parallel Simulated Annealing" (PSA for brevity). Several instances of the simulated annealing algorithm run concurrently, with the temperature schedules for each being determined in part by performance compared to the rest of the group.

With the current popularity of parallel computing architectures, much attention has been directed at reformulating existing algorithms to best exploit concurrent processing. As with many algorithms, there are a variety of ways in which the workload may be shared among a number of processors (Greening, 1990). One can also attempt to directly take advantage of the concurrency (Ram et al., 1996) by using a population of SA algorithms. The most popular approach is to introduce evolution by interleaving iterations of annealing with GA operations such as selection and crossover (Boseniuk et al., 1987; Boseniuk and Ebeling, 1991; Lin and Kao, 1991; Mahfoud and Goldberg, 1995); this is facilitated by the occurrence of the Boltzmann distribution in both the SA acceptance probabilities and as a GA selection operator. GA operators have also been applied to the temperatures (rather than the intermediate solutions) of a population of annealing algorithms; experimental results for GA-controlled-temperature PSA were obtained by Miki et al. (2002).

Earlier (§3), we provided models for several heuristics, but noted that population-based algorithms such as GAs did not fit well into the model. While the transformation describing each generation of a GA satisfies the Markov condition (i.e. transition probabilities are independent of all previous states save the current one), a model must consider every possible population as a separate state. For a population $\mathcal{P}$ distributed over a search space $\mathcal{S}$, there are $\binom{|\mathcal{S}|+|\mathcal{P}|-1}{|\mathcal{P}|}$ possible populations (§A.3), and thus the size of the state vector for the model grows very quickly with the population size. A model for this algorithm must take some shortcut to avoid this state space growth; our choice is to assume that the distribution of states occupied by annealers within the population is equal to the average of the ensemble of all possible distributions.

6.1 Motivation

Although the PSA algorithm is in many ways simpler than a GA, we hope to take advantage of concurrent exploration in a similar way, allowing individuals to be guided by the rest of the population. The advantage over a series of consecutive runs is that any statistic measured for an individual need not be absolute, but can be described relative to the population average. Such statistics have some invariance against simple transformations of the cost landscape; an individual's cost, rate of accepted transitions, or time since last improvement can all be described relative to (and normalised against) the population average for that statistic. This could potentially produce a specification of behaviour which is incorporates less *a priori* information from the search space, instead gathering it as the algorithm runs.

6.2 Model

We shall model a population of "annealers". Each annealer explores the search space independently, but has a temperature determined partially by the performance relative to that of the rest of the population. Specifically, the temperature for each annealer varies according to the difference between that annealer's cost and the average cost for the whole population, with this difference being scaled by the standard deviation of the cost across the population. Formally, if there are n annealers, with costs $c_1 \dots c_n$, then the temperatures $T_1 \dots T_n$ are given by

$$(T_i(t))^{-1} = \beta(t) + \alpha(t)\frac{c_i - \langle c \rangle}{\sqrt{\langle c^2 \rangle - \langle c \rangle^2}}$$

where the averages are taken over the costs of all members of the population. This expression is indeterminate when the standard deviation is zero, i.e. all members have identical cost[1]; when this occurs, we replace the second term with zero.

The parameter α controls the degree to which the individual temperatures depend on the average population cost. A positive α will cause the temperature to decrease for the better members of the population, while a negative value will cause an increase. The magnitude of the change is scaled by the standard deviation of costs of the annealers. In the case where $\alpha = 0$, the temperature depends only on β, and the annealers behave completely independently.

The state of the PSA algorithm at any one time is exactly described by the population; therefore the state vector for the markov model should represent a probability distribution defined over all possible populations, an object we shall refer to as an *ensemble.*

[1]This of course includes the particular case when $n = 1$.

The obvious difficulty is that the information content of the ensemble grows very rapidly with population size, making direct modelling impractical.

To solve this problem, we take an approximation. We produce an "ensemble average" population, where the number of members at each search point is averaged across all possible ensembles. This average population is a small enough structure to used as the state for the model; in fact, we combine the averaging with the PSA algorithm, so that each iteration transforms one averaged population to another. We do not model finite population sampling effects; these leave the average population unchanged, and could therefore only be introduced if we were to expand the model state to include more information from the ensemble. Note also that we do not need the averaged population to have integral numbers of members at each search point; we therefore normalise the values, so that our model transforms a probability distribution rather than a population.

6.3 Results

Due to the nonlinear dependence of the temperature T on the schedule parameters α, β, we are unable to produce a closed-form expression for the search cost in terms of the parameters, and we cannot compute the gradient efficiently. Cost evaluation must instead be performed by repeated transformation of the initial distribution $\mathbf{p}(0)$, and the search for optimal parameters is conducted using a non-gradient method, namely the Nelder-Mead Simplex Algorithm.

WYA-optimised α and β schedules were produced for both a 16-bit hurdle problem and a 20-variable Max-SAT problem. Populations were initialised as sets of random points from the search space (i.e. distributed in proportion to the state sizes). The β schedule for the former (fig. 6.1) is indistinguishable from a "normal" hurdle schedule. The corresponding α schedule (fig. 6.2) is near-uniform for the most part, with a sharp increase at the last iteration; doing so reduces the final-iteration temperature for the best members of the population. Note that while these members may be overtaken by those of higher cost on the final iteration, this can only happen if the higher-cost members take a downhill transition, and thus a higher final-iteration temperature is of no benefit to any member of the population (this is why the β schedule still has a drop in temperature for the final iteration).

The schedules shown in figs. 6.3, 6.4 are those produced for a Max-SAT problem. While β schedule retains a few of the features observed in §5.4 (such as the "cold start", and the general range of temperatures), the schedule rapidly departs from the norm around half way through, with a sudden decrease to (approximately) the initial temperature, and then a slow increase for the rest of the search. At the same time, the α schedule has a sudden increase, giving a larger difference in temperature between the high- and low-cost

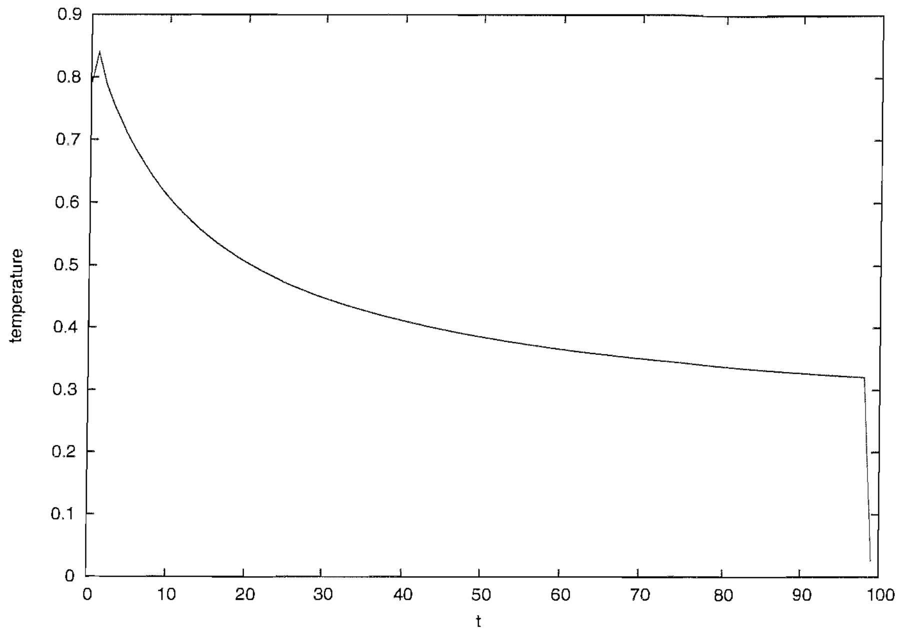

FIGURE 6.1: Optimised $\beta(t)^{-1}$ for hurdle problem

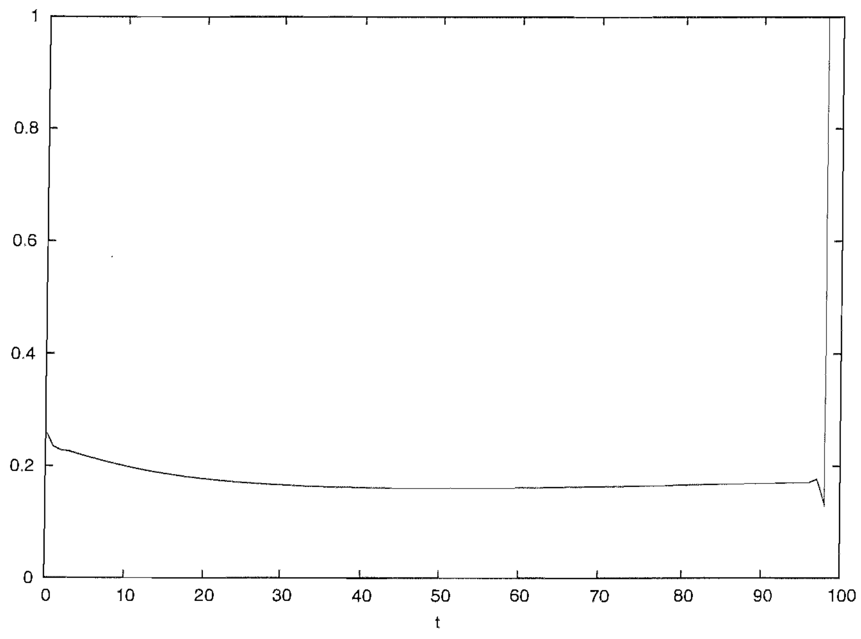

FIGURE 6.2: Optimised $\alpha(t)$ for hurdle problem

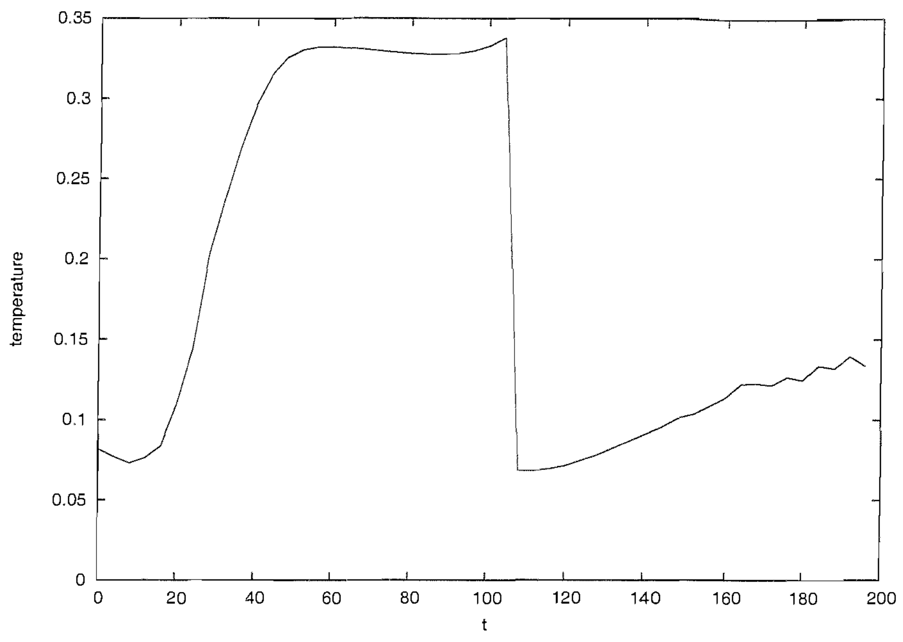

FIGURE 6.3: Optimised $\beta(t)^{-1}$ for Max-SAT problem

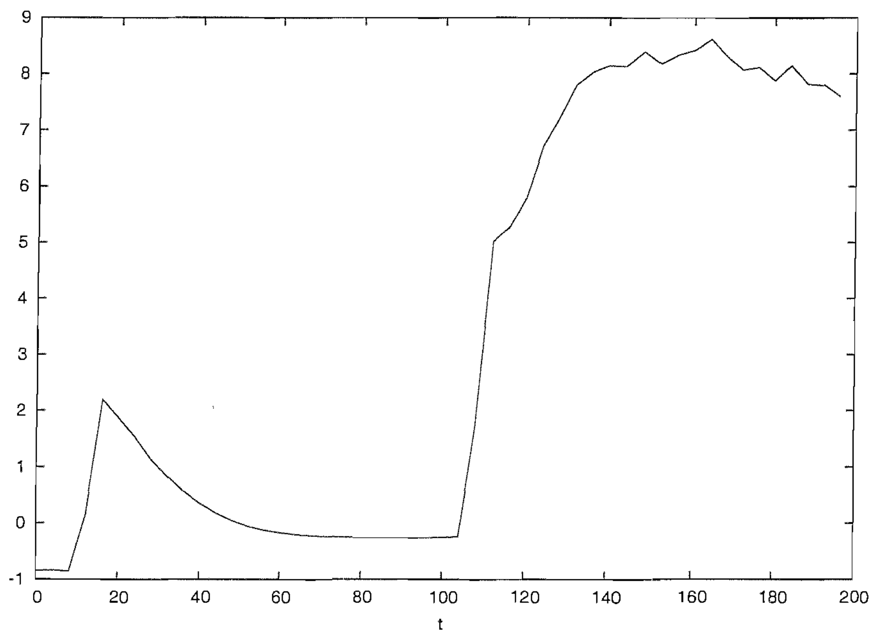

FIGURE 6.4: Optimised $\alpha(t)$ for Max-SAT problem

population members. The effect on the population should be that the best individuals are refused uphill transitions, while the worst are permitted much more freedom.

Figs. 6.5 and 6.6 show the average costs (plotted against time) predicted by the PSA model. These are compared against results obtained by using the optimised schedules in a PSA algorithm, averaged over 20 000 runs. For populations of size 5 and 100, the cost of the best member of the population is shown. Also plotted is the equivalent result

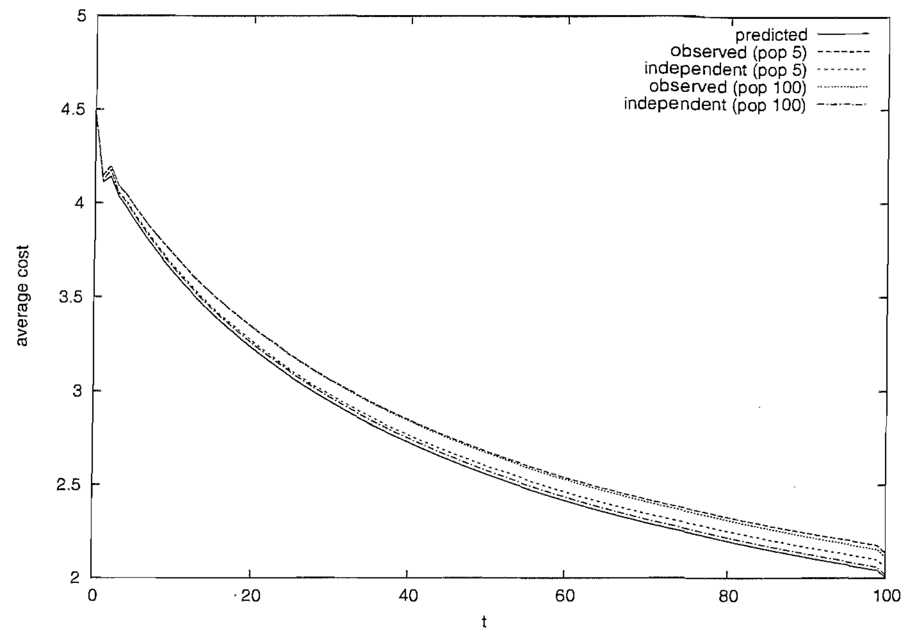

FIGURE 6.5: Predicted and actual performance of optimised PSA schedules on a 16-bit hurdle problem.

for a population of "independent" annealers with no communication, i.e. $\forall t, \alpha(t) = 0$. Unfortunately, it appears that this independent population (equivalent to annealing a number of times and taking the best result) produced a lower cost than PSA; the effect of adding (this form of) communication between the annealers was to retard their progress. This is particularly evident in the case of the Max-SAT results, where the cost begins to rise shortly after the schedules depart from (an approximation to) their non-PSA equivalents.

The predictions made by the model are of course influenced by the same factors which affected our previous optimised schedules on barrier tree models (§5.3); here we have an additional contribution to error. The model assumes a very large population, while the results were obviously produced with finite and relatively small populations. Based on this, and our earlier observations that the predicted costs were consistently lower than actual costs, it is unsurprising that here also the predictions provide lower bounds for the actual results.

In fig. 6.7, the minimum cost in the population is plotted against time, instead of the average cost. The minimum is clearly of greater practical importance, but the average is simpler to model. The relationship between the two is somewhat analogous to that between BSF and WYA cost, and indeed we have reason to believe that schedules producing a good average cost will also produce a reasonably good minimum cost, simply because the average is an upper bound for the minimum.

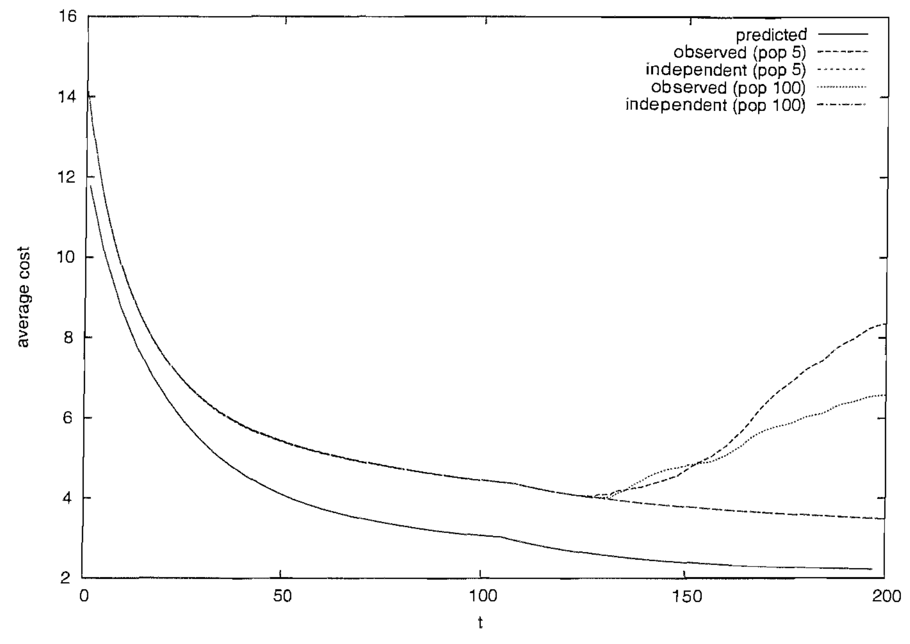

FIGURE 6.6: Predicted and actual performance of optimised PSA schedules on a 20-variable Max-SAT problem.

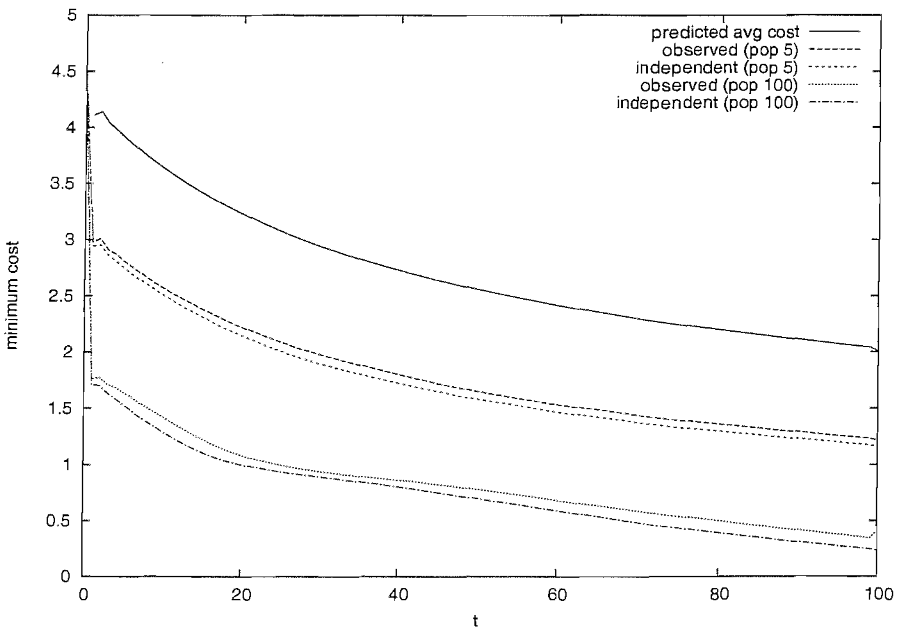

FIGURE 6.7: Minimum cost from a PSA population using the schedules of figs. 6.1, 6.2 on a 20-variable hurdle problem.

We should note that the optimised schedules for the Max-SAT problem both show some irregularity towards the end. This is a little unusual for a WYA schedule, as the end of the schedule usually has the largest effect on the cost; one would expect to find noise nearer the beginning of the schedule, where the effect of a small perturbation upon the final cost is likely to be minimal. A possible explanation is that the length of the schedule and the advantage from multiple annealers is such that the model predicts near-certain convergence on the global optimum long before the end of the schedule, although the cost predicted by the model does not appear to support this. It may be that there is a sudden change in strategy, with the first part of the schedule designed to reach a distribution from which a descent (the second part) is most likely to reach the optimum. This would make the final cost less sensitive to the later parts of the schedules, because all that matters is that the probability of an uphill move is negligible. It may well be that the temperature is sufficiently low, and the probability of the population being in a minimum is so high, that the effect of these fluctuations in temperature is insignificant. The fact that these fluctuations appear some time after the sudden drop in β^{-1} supports this explanation, as does their appearing slightly sooner in the α schedule; as the population converges, the variance reduces, and thus the effect of α decreases.

6.4 Conclusions

We have seen already that schedules optimised for the PSA model do not produce good results when used with the "real" PSA algorithm. The differences between the barrier tree models and the actual search spaces do not provide adequate explanation, as these differences should be absent in the case of the hurdle problem. That the optimisation of the model selects schedule pairs *worse* than the trivial "independent" schedules suggests that an important detail is missing from the model.

The most likely cause is the fact that the model equates the probability distribution to the average of the ensemble of distributions, discarding all other information about the ensemble; approximations of this kind are necessary for populations of nontrivial size, in order that the state space of the model should not become too large. A better representation of the true behaviour might be possible if the model were to include further statistics, such as the variance either within the population, or of the distribution of populations in ensemble space.

Where we currently optimise schedules to minimise the average cost in the population, it would be useful to instead minimise the minimum cost, i.e. to measure the cost of a population as that of its lowest-cost member. In the large population limit, this minimum cost approaches the minimum for the whole search space, thus the results would only be meaningful if computed for a more detailed model, such as described above, parameterised by the population size.

The foundation stone of the PSA algorithm is the assumption that communication between annealers, in the form we prescribed, can be used to improve the search significantly. It may be the case that the cost of one annealer relative to the average is not a good indicator of the best behaviour. If one annealer is at higher cost than the rest of the population, then it could be stuck in a local minimum, or it could be searching for a descent; in one case the temperature should be raised, in the other it should be lowered. Restricting the inverse temperature to a linear function of the (normalised) cost difference also constrains the behaviour: an adjustment to help the worst part of the population cannot be made without also applying the opposite adjustment to the best part.

Some improvement may perhaps be gained by adjusting the temperature according to other factors. The proportions of downhill, uphill and rejected transitions in the recent history of the annealer could be used. It seems reasonable that an annealer with a large proportion of rejected transitions is both in a minimum, and has temperature too low to escape, while one with many recent downhill transitions should probably continue moving downhill. While any of these measurements could be used to control a single annealer, adjusting them according to the population average gives a way to automatically calibrate them according to the landscape, specifically that part of the landscape which the population currently occupies.

An interesting future experiment would be to look at, or even attempt to control, the spatial correlation between the annealers on the landscape. In comparison to a conventional GA, this algorithm has no crossover, but also lacks any other mechanism for replacing poorer individuals (e.g. replication of the best through selection and sampling). An annealer which finds a better region to explore has no way to encourage others to join it. On many landscapes, a distance metric can be specified in terms of the number of mutations (i.e. transitions to a neighbour) required to travel from one point to another. It may be of some use to attempt to control the degree of spatial correlation by adjusting acceptance probabilities according to the position of the annealer, and whether the proposed transition moves it toward or away from the rest of the population (i.e. reduces or increases correlation). The goal would be to reduce duplication within the population, and encourage movement towards better regions.

Chapter 7

Optimal Heuristics

So far, we have been mostly concerned with producing optimal parameters for heuristics, with the focus being on understanding and improving the search. In this chapter we take a new stance, and produce *optimal heuristics* as a means to study the difficulty of our problem instances.

The difficulty of a problem has traditionally been measured in terms of the time taken to reach a solution. For discrete, finite search spaces such as ours, this idea can be formalised easily as the *average first passage time* (§7.2), i.e. the number of steps taken to reach the global optimum from a randomly chosen starting point.

We generalise the decision process (i.e. the acceptance probabilities) of simulated annealing to cover arbitrary functions of the "to" and "from" states (§7.3); we then find the assignment of acceptance probabilities which minimises the average first passage time (§7.2). Effectively, the heuristic has full information about the search space, but is only able to control its own movement by accepting or rejecting randomly proposed neighbours. For comparison, the experiment is repeated in §7.4 with the heuristic having access only to cost (rather than exact state) information, and also with single-parameter variants of simulated annealing and variable mutation search (§7.5). Results and discussion for the four algorithms follow in §7.6.

7.1 Introduction

Simulated annealing, variable mutation search and descent are members of a class of algorithms which have a similar underlying structure: each consists of a series of distinct iterations, between each of which the only information preserved is the current position of the search. Within each iteration, one neighbouring search point is examined, and accepted with a probability dependent on the costs of the old and new points, and the search parameters.

As we saw earlier (§3), each iteration of an algorithm of this form may be represented by a transition matrix, computed from the costs of the states and the search parameters. We now take this one step further, and allow the acceptance matrix to contain arbitrary values in $[0,1]$. Since any algorithm of the iterated select/accept form described above may be represented in this way, the set of all such matrices encompasses all possible algorithms of that form. Considering the entries of these matrices as the parameter set for the search, we may optimise across all possible series of acceptance matrices, with the result being a series of matrices representing the optimum heuristic for the problem (again, subject to the restrictions described above).

An obvious practical difficulty is that the number of parameters involved is enormous; if there are n states and T iterations, then there are $n(n-1)T$ independent parameters to optimise. This problem is largely avoided when FPT cost (average number of iterations before the optimum is reached) is used. Associated with WYA and BSF cost was an implicit dependency on "time"; because the number of iterations was limited, any choices made by the heuristics were influenced by the number of iterations remaining. This is because the goal of the heuristic (to use the next $(T-t)$ iterations in such a way as to minimise the final search cost) is itself parameterised by the current time t. FPT cost differs in this respect, as the task presented to the heuristic at each iteration (to reach the global optimum as soon as possible) is identical.

The result is that the transition matrix for an FPT-optimal heuristic will be time-independent[1], and thus only $n(n-1)$ parameters need be optimised. It is also worth noting that FPT optimisation for a time-dependent heuristic would be considerably more difficult, and only possible for specific types of time-dependence; we give a little more thought to this matter in §7.5.

FPT is in some sense a more "absolute" measure than BSF or WYA cost; the absence of a controlling parameter (such as the number of iterations) allows for cleaner comparisons between search heuristics. Given a search algorithm with optimised parameters (or described without any parameters), the FPTs for various problems can give hints about their relative difficulty.

7.2 First Passage Time

The average "first passage time" T_{fp} is the average number of iterations before the optimum is reached for the first time. For any algorithm where entering a particular state causes the global minima to become unreachable, T_{fp} is infinite. The simplest

[1]By "FPT-optimal heuristic", we mean a heuristic where the transition probabilities may be chosen freely, and have been done so to minimise FPT. When a parameterisation is used which constrains the heuristic (in the case of simulated annealing, for example), the optimal parameters may be time-dependent.

example of this is descent, which is unable to reach any global minima after entering the basin associated with a local minimum. The criterion that the global minimum should always be reachable is a weaker constraint than ergodicity (which requires that *all* states be reachable from all others), and is necessary but *not* sufficient (§A.4) for T_{fp} to exist.

Average first passage time is most easily calculated by modifying the transition matrices in a similar way to the method used for BSF cost (Boese and Kahng (1994) and §3.2.2). When calculating BSF cost, we produced a transition matrix which "trapped" costs below a certain threshold by replacing the appropriate columns of the matrix with those of the identity; the probability of having ever entered at least one of those states was given by the sum of the probabilities corresponding to the trapped states. For first passage time we shall use a similar method, with the matrix columns corresponding to the global minima being replaced with zeros. The transition matrix is of course no longer stochastic, with the probabilities corresponding to transitions *from* the global minima being "leaked" at each iteration.

7.2.1 Derivation

Let $\mathbf{e}$ be a vector of values in $\{0, 1\}$, with each $e_i = [\forall j, c_i \leq c_j]$ (with j indexing all states), i.e. the elements of $\mathbf{e}$ corresponding to global minimum states are set to one, all others are zero. Then the modified transition matrix $\hat{w}$ is given by $\hat{w}_{ij} = (1 - e_j)w_{ij}$.

Suppose $\hat{w}$ is applied n times to an initial vector $\mathbf{p}(0)$, to produce $\mathbf{p}(1) \ldots \mathbf{p}(n)$. Each element $p_i(t)$ is a sum of the probabilities of all trajectories of length t from state j to state i, weighted by $p_j(0)$. All trajectories which visit a global minimum state have probability zero, and thus $p_i(t)$ is the probability of being at state i, given that no global minimum has ever been visited. The probability that no global minimum has been visited thus far must then be the sum over i of these values, so we have

$$\begin{aligned}
\mathbb{P}(T_{fp} \geq t) &= \mathbf{1}^\top \hat{w}^t \mathbf{p}(0) \\
\mathbb{P}(T_{fp} = t) &= \mathbb{P}(T_{fp} \geq t) - \mathbb{P}(T_{fp} \geq t + 1) \\
&= \mathbf{1}^\top \left(\hat{w}^t - \hat{w}^{t+1}\right) \mathbf{p}(0) \\
&= \mathbf{1}^\top (I - \hat{w})\, \hat{w}^t \mathrm{p}(0).
\end{aligned}$$

Note that $\mathbf{1}^\top (I - \hat{w})$ is identical to $\mathbf{e}^\top$; this makes sense, as it is the probabilities corresponding to the global optimum states which disappear on every iteration, and premultiplying by $\mathbf{e}^\top$ effectively sums over those states.

Expressing the average first passage time in terms of these probabilities, we have

$$\begin{aligned}\langle T_{fp}\rangle &= \sum_{t=0}^{\infty} t\mathbb{P}\left(T_{fp}=t\right)\\ &= \sum_{t=0}^{\infty}(t+1)\,\mathbb{P}\left(T_{fp}=t\right)-\sum_{t=0}^{\infty}\mathbb{P}\left(T_{fp}=t\right)\\ &= \mathbf{1}^{\mathsf{T}}\left(I-\hat{w}\right)\left(\sum_{t=0}^{\infty}(t+1)\,\hat{w}^{t}\right)\mathbf{p}(0)-1\end{aligned}$$

where the identity $\sum_{t=0}^{\infty}\mathbb{P}\left(T_{fp}=t\right)=1$ arises from conservation of probability. The summation may be simplified by expressing the terms as derivatives of powers of $\hat{w}$:

$$\begin{aligned}\sum_{t=0}^{\infty}(t+1)\,\hat{w}^{t} &= \sum_{t=0}^{\infty}\frac{\mathrm{d}}{\mathrm{d}\hat{w}}\hat{w}^{t+1}\\ &= \frac{\mathrm{d}}{\mathrm{d}\hat{w}}\sum_{t=0}^{\infty}\hat{w}^{t+1}\\ &= \frac{\mathrm{d}}{\mathrm{d}\hat{w}}\left(I-\hat{w}\right)^{-1}\end{aligned}$$

which simplifies[2] to $\left(I-\hat{w}\right)^{-2}$, leaving us with

$$\begin{aligned}\langle T_{fp}\rangle &= \mathbf{1}^{\mathsf{T}}\left(I-\hat{w}\right)\left(I-\hat{w}\right)^{-2}\mathbf{p}(0)-1\\ &= \mathbf{1}^{\mathsf{T}}\left(I-\hat{w}\right)^{-1}\mathbf{p}(0)-1.\end{aligned}$$

If $\hat{w}$ is dependent on some parameter β, then the gradient of the FPT is given by

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}\langle T_{fp}\rangle &= \mathbf{1}^{\mathsf{T}}\frac{\mathrm{d}}{\mathrm{d}\beta}\left(I-\hat{w}\right)^{-1}\mathbf{p}(0)\\ &= \mathbf{1}^{\mathsf{T}}\left(I-\hat{w}\right)^{-1}\frac{\mathrm{d}\hat{w}}{\mathrm{d}\beta}\left(I-\hat{w}\right)^{-1}\mathbf{p}(0).\end{aligned}$$

7.2.2 Numerical Instability in Optimisation

Optimisation of average FPT cost is hampered somewhat by a numerical instability arising from the frequent near-singularity of the matrix expression $(I-\hat{w})$. Evaluation of FPT cost requires the inverse of this matrix, and is therefore sensitive to any noise introduced from the inversion. The gradient is affected to a greater extent, as this inverse appears twice in the product, thus multiplying the error.

The benefit to the speed of optimisation attained through using gradient information must be weighed against the chance of the search failing to converge due to an inaccurate gradient. Our decision is influenced mostly by the number of parameters involved in the

[2]Since $0=\frac{\mathrm{d}AA^{-1}}{\mathrm{d}x}=A\frac{\mathrm{d}A^{-1}}{\mathrm{d}x}+\frac{\mathrm{d}A}{\mathrm{d}x}A^{-1}$, we have $\frac{\mathrm{d}A^{-1}}{\mathrm{d}x}=-A^{-1}\frac{\mathrm{d}A}{\mathrm{d}x}A^{-1}$.

optimisation; for algorithms involving only a single parameter, we elect to discard the gradient information and instead use the Nelder-Mead Simplex Algorithm (although this is not the ideal choice, the 1D case is not too difficult to optimise anyway). Where a large number of parameters are involved (§7.4), we retain this information and use the Scaled Conjugate Gradients algorithm.

Difficulties in inverting $(I - \hat{w})$ can be sidestepped to some extent by optimising on approximations to this matrix. We substitute $(I - (1 + \delta)\hat{w})$ for this expression, with $|\delta| \ll 1$. The optimum parameters may now be reached through a series of optimisations with successively smaller values of δ, with the exact result being produced when $\delta = 0$. The advantage of this approach is that the search is more stable for the earlier iterations, as the error from matrix inversion is traded for inaccuracy in the finished result. Each time δ is reduced, the search begins anew from the result of the previous iteration; the error is increased, but the distance to the new optimum is small. Although eventually the error will likely become large enough to make this improvement insignificant, the iterative optimisation fails more gracefully than a direct approach, as the result can be taken to be the best point reached before the optimisation begins to fail, instead of the best point reached during a failed optimisation run.

7.3 Lower Bound Algorithm

For a given problem, the set of all possible search heuristics (with the "select/accept" form described above) has a global optimum, the average FPT of which is a lower bound for that of all other heuristics in the set. This heuristic effectively has full *a priori* knowledge of the search space, and is fully aware of its own state, insofar as it can choose behaviour based on which search point it is at, rather than just the cost at that point.

The FPT for this algorithm is not hampered by "deceptive" features of the landscape such as local minima, and thus it may be used as measure of the intrinsic difficulty of *traversing* a search space, from a random point to a global minimum.

7.3.1 Definition

Formally, the "lower bound algorithm" for any problem is, of those heuristics which satisfy the following constraints, the one which produces the minimum FPT:

- The state of the heuristic must be representable as a single point in the search space (this precludes the use of any history information)
- Once per iteration, a randomly-chosen neighbour of the current position may be examined, and optionally adopted as the new position

- Only the neighbour itself is observable; direction information may not be used

(Note that optimality for this heuristic is defined in terms of the search space presented to the heuristic, not of any underlying space of which it may be an abstraction. For example, when applied to a barrier tree model of a problem, only the performance on the model is of concern.)

Thus each iteration shall consist of a neighbour being proposed, and optionally selected, with no other useful actions being possible. Any such algorithm is entirely specified by a decision function $\mathbb{P}_A(i \to j)$, giving the probability of accepting a transition from state i to state j. Note that having full information about the search space and the current position removes any advantage associated with time-dependent heuristics and randomisation. The current position is known precisely, and thus there is no need to use "time" to make inferences about the current state, nor to use randomised behaviour to compensate for uncertainty.

Our $\mathbb{P}_A(i \to j)$ will therefore only ever take values from $\{0, 1\}$, and our "lower bound algorithm" may be reduced to an acceptance matrix, similar to those used for the heuristics in §3.1. The optimal acceptance matrix is found using Dijkstra's algorithm for the shortest path to the optimal state, with a small complication in the calculation of distance labels. Helpfully, this method also simultaneously computes the average FPT from each state; the FPT from a random starting state can then be calculated trivially.

7.3.2 Minimising FPT Using Dijkstra's Algorithm

The problem of choosing the acceptance probabilities so as to minimise the average first passage time is similar to that of finding shortest paths on a digraph. Representing the states of our model as graph nodes, and labelling edges according to transition probability, we must find paths from each state to a global minimum, minimising a value which is a function of the edges traversed.

However, FPT differs from the usual distance measure associated with graph pathfinding problems, as the FPT from some node to the optimum is not determined solely by that of the "best" adjoining node and the weight of the associated edge; the FPT depends on that of all neighbouring nodes to which transitions are *acceptable* (i.e. have an acceptance probability of 1) from the starting node. Although it may at first seem reasonable that only one neighbour should ever be acceptable to a node, this is only the case if neighbours are to be freely chosen. In reality, the fact that neighbours are proposed in a random order means that there is a probabilistic cost associated with waiting for a transition to any particular neighbour. Should a neighbour with FPT less than that of the current node be suggested, it should always be accepted.

The FPT from state i is given by

$$\tau(i) = [\exists j, c_j < c_i] \left(\langle \tau_{wait}(i) \rangle + \langle \tau_{next}(i) \rangle \right)$$

where $\tau_{wait}(i)$ and $\tau_{next}(i)$ are respectively the time spent waiting for an acceptable transition, and the FPT for the new node after an accepted transition. For nodes corresponding to global minima, τ is zero, otherwise it is the sum of these two times.

Let the transition probabilities from node i to j be given by m_{ji}, and the acceptance probabilities by A_{ji}. If $a(i)$ is the probability of accepting a randomly (according to m_{ji}) chosen neighbour of state i, so that $a(i) = \sum_{j=1}^{n} A_{ji} m_{ji}$, then the distribution of $\tau_{wait}(i)$ is given by $\mathbb{P}\left(\tau_{wait}(i) = t\right) = a(i)\left(1 - a(i)\right)^{t-1}$, which is a geometric distribution with mean $a(i)^{-1}$.

The time $\tau_{next}(i)$ is given by an average over the acceptable nodes, weighted by their (normalised) transition probability:

$$\tau_{next}(i) = a(i)^{-1} \sum_{j=1}^{n} A_{ji} m_{ji} \tau(j).$$

Thus for states other than the global minima, we have

$$\tau(i) = a(i)^{-1} \left(1 + \sum_{j=1}^{n} A_{ji} m_{ji} \tau(j) \right)$$

and

$$T_{fp} = \sum_{i=1}^{n} \tau(i) p(0)_i.$$

The optimal strategy could in theory be found through exhaustive search over all possible acceptance matrices. However, this is very inefficient (and indeed impractical) as it is possible to shortcut this process with a little thought.

Any acceptance strategy implicitly assigns a first passage time to each node, thus defining an ordering over the nodes. The optimal strategy has the property that each node must accept transitions to a neighbour with lower FPT, and must refuse neighbours with higher FPT. If a so-called "optimal" strategy required rejection of a lower-FPT neighbour, or acceptance of a higher-FPT neighbour, then the strategy could be improved by reversing that preference (the FPT for the current node would decrease, as would that of all higher-FPT nodes), and thus the original strategy could not have been optimal. The behaviour for equal-FPT nodes does not matter, and values of A_{ji} for which $m_{ji} = 0$ are likewise irrelevant. Effectively, the acceptance matrix for (a strategy equivalent to) the optimal strategy has the form

$$A = P \begin{bmatrix} 0 & 1 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & \\ 0 & 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & \\ \vdots & & \vdots & & \ddots \end{bmatrix} P^{-1}$$

where P is a permutation matrix (i.e. in every row or column of P, there is a single 1 and $n-1$ zeros).

For any node, we can find the minimum FPT assigned to it by any possible strategy; the set of minimal FPTs for the nodes is an intrinsic, strategy-independent property of the problem. We use the ordering induced by these FPTs to construct a strategy which assigns these intrinsic minimal FPTs to every node, thus guaranteeing optimality.

We construct the strategy progressively. Suppose we have an optimal strategy on the first k nodes (counting from the global minima, with FPT increasing). The FPT for each of these nodes is independent of nodes $(k+1)\ldots n$, all of which have higher FPT and are therefore not acceptable to the lower nodes. We take each of the $n-k$ remaining nodes in turn, and calculate the FPT assuming only the first k nodes are acceptable; the $(k+1)$th node is the one which to which the smallest FPT is assigned. To see why this is valid, consider the possibility that this selection is incorrect; suppose there exists another node which, with an appropriate choice of acceptance rules, could have lower cost. This would require that the FPT of the other node could be improved in some way by increasing the set of accepted nodes, which would require that there exist another node *between* that and the kth, precluding that from being the $(k+1)$th. All that remains is to produce a starting point for the algorithm, consisting of a set of points and an optimum acceptance matrix; the efficient choice is the set of global minima (for which the acceptance matrix is irrelevant), the elegant choice is the empty set.

This algorithm is specified formally in alg. 4; it has running time $O\left(n^3\right)$ if implemented exactly as specified, or $O\left(n^2\right)$ if optimised to adjust FPT labels when a node is fixated, rather than calculating them anew. A little care may be necessary to avoid division by zero; this happens when no neighbours of a node are fixated (because the transition probabilities to the fixated nodes are all zero). Treating the result of the division as infinity is suitable, if the numerical representation permits this, otherwise an arbitrary FPT may be assigned so long as the node is marked ineligible for fixation.

Algorithm 4: Dijkstra's algorithm modified for FPT optimisation

```
input  : $m, \mathbf{c}, n$: neighbourhood matrix, state costs, number of states
output: $A, \tau$: Optimal acceptance matrix and first passage times
/* Begin with optimum states fixated, with FPT of 0                    */
$fixed \leftarrow \{i : \forall j, c_i \leq c_j\}$;
$remaining \leftarrow \{1 \dots n\} \setminus fixed$;
$\forall i, j, A_{ji} \leftarrow 0$;
$\forall i, \tau_i \leftarrow 0$;
while $remaining \neq \emptyset$ do
    /* Update upper bounds for FPT. Efficiency could be improved by
       updating the previous FPTs to reflect the newly-fixated nodes,
       rather than starting from scratch.                              */
    foreach $i \in remaining$ do
        $\tau_i \leftarrow 0$;
        total $\leftarrow 0$;
        foreach $j \in fixed$ do
            $A_{ji} \leftarrow 1$;
            total $\leftarrow$ total $+ m_{ji}$;
            $\tau_i \leftarrow \tau_i + m_{ji}\tau_j$;
        end
        $\tau_i \leftarrow (\tau_i + 1)/$ total;
    end
    /* Of the remaining nodes, fixate those with the lowest FPT bound  */
    $best \leftarrow \{i : j \in remaining \implies \tau_i \leq \tau_j\}$;
    $fixed \leftarrow fixed \cup best$ ;
    $remaining \leftarrow remaining \setminus best$ ;
end
return $(A, \tau)$;
```

7.4 Cost-Dependent Algorithm

Our "lower bound" algorithm is just that - an algorithm whose performance should be a lower bound for the set of heuristics meeting the criteria in 7.3.1. However, these criteria allow access to information not used by simulated annealing; where our lower bound algorithm is free to make an acceptance decision based on the originating and neighbouring state, SA is only able to use the costs of these states[3]. We fully expect that the lower bound algorithm should produce lower FPTs than SA or similar heuristics, but it would be useful to know how much of the difference may be attributed to lack of ability to identify states, and how much is due to the other constraints on the acceptance probabilities.

Let a *cost-dependent algorithm* be an algorithm which operates within the criteria of 7.3.1, but with the further restriction that the identity of any state is hidden from the algorithm; only the cost is visible. Again, the algorithm may be described entirely by an

[3]Of course, SA is further restricted, as (for "uphill" transitions) the log of the acceptance probabilities must be linear in the cost difference.

acceptance matrix, this time with rows and columns labelled by costs instead of states. This matrix will usually be smaller, with size given no longer by the number of states, but by the number of distinct costs those states possess. For some classes of problems, there may be no states (or a very small proportion) with equal cost, in which case the cost-dependence criterion provides no restriction, and the optimal cost-dependent algorithm is simply the lower bound algorithm.

For our test problems (i.e. those described in §4) however, costs are frequently repeated, and this affects the optimal strategy: if we are presented with a randomly chosen neighbour, about which we know only the cost, then it may be one of several possible states, and we are unable to tell whether it leads to the global optimum (and should be accepted) or leads to a local minimum (and should be rejected). When equal-cost states can no longer be distinguished, the algorithm must accept the possibility that an incorrect decision could be made. Our earlier assumptions about time-independence and determinism no longer hold: a heuristic with no variation in behaviour is more likely to repeat the same mistakes and stay trapped in the same region of the space. We now permit noninteger probabilities, but to simplify the calculations we assume that the optimal heuristic is still time-independent.

We have no way to construct algorithmically the acceptance matrix for the optimal cost-dependent algorithm, and we must instead treat it as a matrix of parameters over which the FPT must be optimised. On account of the high dimensionality of the search space, we use the Scaled Conjugate Gradient algorithm to do so.

By comparison to the lower bound algorithm and to simulated annealing, we intend to determine (in §7.6) whether it is the loss of information about individual states, or the lack of freedom in choosing the acceptance probabilities, that is of greater detriment to the heuristic's performance.

7.5 Single-Parameter Heuristics

Finally, we will compute first-passage times for simulated annealing and variable mutation. We will use constant parameter schedules for each, referring to the algorithms as "constant-temperature annealing" and "fixed-rate mutation" respectively. As with the cost-dependent algorithm, there is no reason to expect that the optimum annealing/mutation schedule should be independent of time, but we again make this approximating assumption so that the optimisation remains feasible.

It is of course possible to describe non-terminating annealing schedules with finitely many parameters, but to do so in such a way that the FPT may be calculated efficiently is more difficult. As a last resort, one could substitute measurement for explicit calculation, by evaluating the sum

$$\begin{aligned}\langle T_{fp}\rangle &= \sum_{t=0}^{\infty} t\mathbb{P}\left(T_{fp}=t\right)\\ &\approx \sum_{t=0}^{T_{max}} \mathbf{1}^{\top}\left(I-\hat{w}(t+1)\right)\left(\prod_{i=1}^{t}\hat{w}(i)\right)\mathbf{p}(0)\end{aligned}$$

for sufficiently large T_{max}. This method was in fact used to verify the results of §7.2.1. This calculation requires $O\left(T_{max}\right)$ multiplications of a vector by a matrix (the sum can be accumulated as the product is calculated), compared to the normal constant-time method. Difficulty arises when this less efficient computation is combined with a search space of increased dimensionality (due to a larger number of parameters); not only are there many more iterations required for optimisation, but the time taken to perform each iteration is increased too.

Furthermore, a parameterisation which retains reasonable computational efficiency is not enough; one must also design it in such a way as to minimise the impact of the artificial constraints introduced. The problem is surely not intractable, but shall remain an avenue for future work.

We intend to find the optimal annealing temperature and mutation rate for each problem, in order that we may draw some (admittedly simplistic) conclusions from the comparison of the FPTs. The variable mutation heuristic suffers from a somewhat artificial handicap insofar as it will sometimes waste an iteration by considering a self-transition. This possibility arises because the number of mutations applied to generate a neighbour is Poisson-distributed (§2.2.2), and can therefore potentially be zero. It is reasonable to expect that in any sensible implementation of the algorithm, this case will be avoided, with Poisson deviates drawn (a procedure with assumedly insignificant running time) until a nonzero value is produced. The new distribution $\mathbb{P}\left(n|\lambda\right)$ is adjusted by setting $\mathbb{P}\left(0|\lambda\right)=0$, and renormalising the remainder of the distribution:

$$\begin{aligned}\mathbb{P}\left(n|\lambda\right) &= [n>0]\,e^{-\lambda}\frac{\lambda^n}{n!}\left(1-e^{-\lambda}\frac{\lambda^0}{0!}\right)^{-1}\\ &= [n>0]\,e^{-\lambda}\frac{\lambda^n}{n!}\left(\frac{e^{\lambda}-1}{e^{\lambda}}\right)^{-1}\\ &= [n>0]\left(e^{\lambda}-1\right)^{-1}\frac{\lambda^n}{n!}.\end{aligned}$$

The "average mutation matrix" is now

$$
\begin{aligned}
M &= \sum_{n=0}^{\infty} m^n \mathbb{P}(n|\lambda) \\
&= \sum_{n=0}^{\infty} m^n \left(e^\lambda - 1\right)^{-1} \frac{\lambda^n}{n!} - m^0 \left(e^\lambda - 1\right)^{-1} \frac{\lambda^0}{0!} \\
&= \left(e^\lambda - 1\right)^{-1} \left(\sum_{n=0}^{\infty} \frac{(m\lambda)^n}{n!} - I\right) \\
&= \left(e^\lambda - 1\right)^{-1} \left(e^{m\lambda} - I\right).
\end{aligned}
$$

Note that the purpose of this adjustment is not to prevent self-transitions altogether, but just to refuse any decision which is guaranteed to waste an iteration by performing no action. Self-transitions are still possible, either through a series of mutations which produce no net effect, or through a mutation leading to the same state[4].

7.6 Results

Average first passage times were computed for each algorithm on a set of barrier tree problems, and are shown in fig. 7.1. For the single-parameter algorithms, the FPT optimisations were repeated ten times, from different initial conditions, with obvious erroneous results discarded (the optimisation occasionally diverged, producing negative, infinite, or indeterminate FPT). The filtered results were then consistent, usually matching to at least four significant figures.

Single-temperature annealing is a special case of the cost-dependent algorithm, which itself is a special case of the lower bound algorithm. Thus it is no surprise that of these three, the lower bound algorithm is always best, and single-temperature annealing always worst. The mutation search falls outside of this scale, as the ability to move to a point outside the immediate neighbourhood violates the constraints of §7.3. While it is of course possible for an algorithm with this freedom to outperform the others, this particular example is hampered by the inability to make uphill moves; any jump past a barrier *must* take place in a single move.

We might have expected a correlation between FPT and the number of barrier tree states. The problems may be thought of as graphs on which the heuristics perform various types of random walk, and thus the number of states in the tree (i.e. the number of nodes on the graph) would seem to be a reasonable indicator of the difficulty of the problem. At a glance, it appears that no such relationship can be inferred, at least not on the the scale of the problems we examined.

[4]Such mutations are frequent for barrier-tree models, where the expected length of a random walk through the points represented by a state is effectively encoded in the self-transition probability.

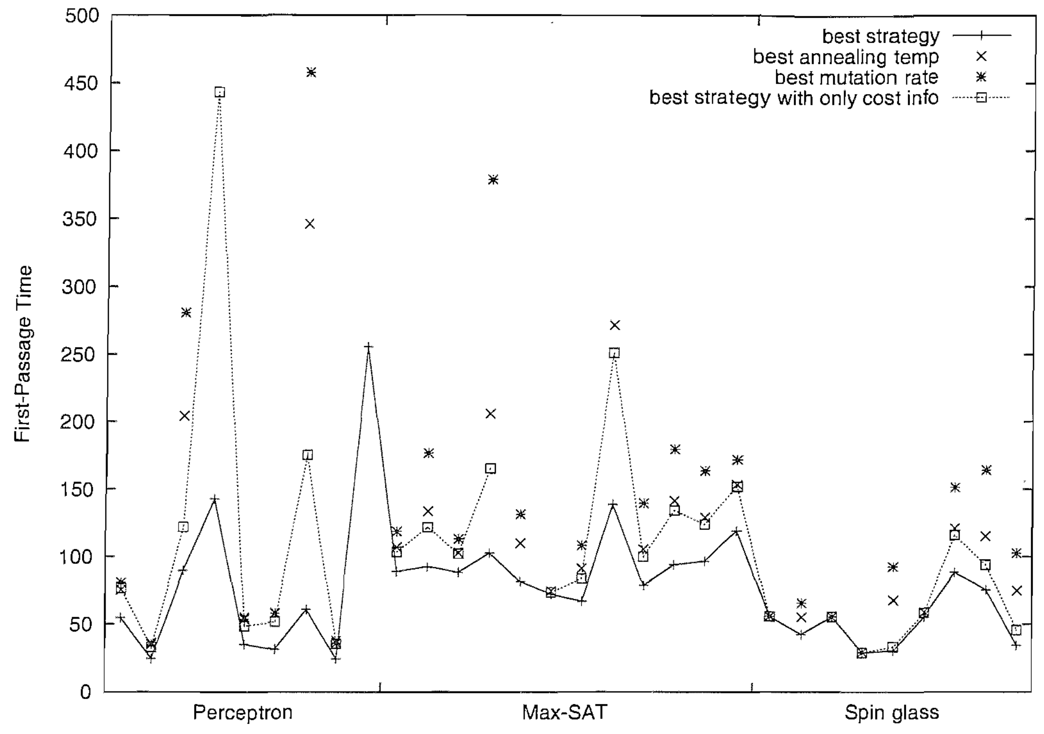

FIGURE 7.1: FPT for four different algorithms on (barrier trees of) a variety of test problems. Within each category, problems are sorted by number of barrier tree states. On a small number of problems, the optimisation for the cost-dependent algorithm consistently diverged, hence the occasional missing points from this series. The other absent data points are those few results with times exceeding 500 iterations.

The difference between the FPT for lower bound and cost-dependent algorithms varies considerably across the problem set. On those problems where the two are identical, one would expect to find that the behaviour of the two algorithms is very similar. In fact, differences between the two are only possible where there exist a number of states with the same cost, in which case the state may still sometimes be uniquely determined by the magnitude of the cost difference of the neighbour selected by the algorithm.

An obvious feature of the results is the apparent superiority of simulated annealing over the mutation search. One can imagine instances of plateau or barrier traversal where constant-temperature annealing would be expected to behave as a random walk (or worse, in the case of the barrier), but where the mutation search could, at least in theory, pass the obstacle in less time through multiple mutations. In fact, we have observed this already for variable mutation search on the hurdle problem (§5.6, Prügel-Bennett (2004b)). However, it may be that in the case of large barriers, the inability of mutation search to move to an intermediate point outweighs the "wandering" tendency of fixed-temperature annealing.

Chapter 8

Conclusions

We have produced and studied a model for the performance of a search, parameterised by the problem instance and the search parameters. Where previous work was limited to modelling very small instances of problems, we use barrier trees as a tool for aggregating search space states, allowing consideration of much larger problems (2^{20} states and above), through a relatively small (typically 20–200) number of model states.

Construction of a barrier tree is possible for any optimisation problem with a finite number of states, although it is most useful for those with discrete cost. As the number of distinct costs increases, the number of barrier tree states will increase, reducing the effectiveness of the state aggregation, although this could be countered by artificially grouping similar costs together. The ability to apply the model to a vast array of problems is a definite strength, although an unfortunate consequence of the generality appears to be a loss of some information about the structure of the space.

8.1 Evaluation of the Model

The most obvious use for such a model is to assist in understanding the connection between the search parameters and the cost produced, with the eventual goal of producing improved search parameters for a problem based on either a brief sampling of the problem, or detailed study of a few representative problems from the same class, the computational cost of which can be amortised across a comparatively large number of problems requiring solution.

However, discrepancies between the predictions of the model and the results obtained through running the search result in model-optimal parameters being substantially sub-optimal on the actual problems. Given that the model is mostly exact, the source of the error is easily identified as the one part which uses an approximation: the aggregation of search points into level-connected sets, and their further amalgamation into level-accessible sets. Where this aggregation was not used, results for the actual problem matched the predictions of the model very closely (§5.2).

Results produced using only the first stage of aggregation (level-connected sets) showed a similar deviation from predictions (end of §5.3), with predicted cost being strictly lower than actual cost. Using the level-connected sets (or any grouping thereof) as the states for a Markov model implicitly assumes that the transitions between these sets possess the Markov property, i.e. that each transition depends only on the current state, and is not otherwise correlated with previous transitions. Of course, we know that this is not the case, but we assume it is so as an approximation, as incorporating history information into the model would cause rapid expansion of the state space.

This assumption is at fault because the "exit point" from a level-connected set is usually correlated with the "entry point". The path through the set of points is in effect a random walk, with a chance of moving to a neighbour from any point on the boundary of the set; neighbouring sets which are spatially close to the entry point are more likely to be chosen. This also applies to self-transitions; where a neighbour is rejected by Simulated Annealing, our model assumes that the next neighbour is chosen independently, whilst the truth is that the rejected neighbour has a greater chance of selection.

Furthermore, the length of time spent within any level-connected set (reflected in the model as a component to the self-transition probability) is itself correlated with the entry point. When the temperature is such that the probability of an uphill transition is significantly reduced, the entry point for a set is most likely to be one of those perimeter points with a large proportion of higher-cost neighbours (as it is more likely that the set is reached via a downhill transition than an uphill one). However, such a site is also likely to have fewer downhill transitions available, reducing the probability that the search leaves via that same point. For similar reasons, the grouping of level-connected sets according to accessibility is also a source of error (§4.3.2).

8.1.1 Possible Refinements to the Search Space Model

It is possible to add further detail to the barrier tree model by partitioning the boundaries of level-accessible sets according to the reachable neighbours, in an effort to preserve some of the structure within the sets, at the cost of an increased number of states. This was attempted in Hallam (2006), with the boundaries of level-accessible sets divided according to whether or not a downhill transition was possible. This barrier tree was used to model descent, but was found to provide only a very small increase in accuracy.

An alternative may be to incorporate limited "history" information into the model. The resulting increase in state size would be undesirable, but might be reasonable if extra "sub-states" were added selectively to those sets with the highest entry-exit correlation. This would not directly improve the SA rejection/self-transition situation described above, as only a (very brief) history of visited sets would be available; previous algorithm choices could not (and for sake of generality should not) be encoded in the state representation. However, our model of the SA heuristic could be adjusted to take advantage of the connection between these sub-states, with the rejection probability for a neighbour contributing to a sub-state specific to that neighbour, rather than a self-transition to the same sub-state.

8.1.2 Population-based Models

Direct models of population-based search methods (such as GAs) are usually impractical, as the state of a population grows exponentially with the population size. In §6 we introduced a "Parallel Simulated Annealing" algorithm, effectively a population-based variant of SA. The approach used in this case was to assume that the population distribution is equal to its own ensemble average, allowing the population to be represented as a single distribution over the search space. This model has several shortcomings which we noted in §6.4, the most significant of which is that there is no adjustment for population size; the model implicitly assumes that the population is large enough that finite-population effects disappear.

The population model used for the PSA algorithm was the simplest possible representation of a population; it is likely that an improved model may be constructed by incorporating more information about the ensemble distribution than just the average. Obviously some balance must be struck between the preservation of information and the size of the model; preserving *all* information about the ensemble would produce too large a model. It is important however to note that the PSA model transformed the state nonlinearly, preventing (or at least adding significant complication to) the calculation of a gradient with respect to the search parameters; it is likely that models of other population-based searches will be similar in this regard, greatly increasing the time required for parameter optimisation.

A population with recombination (such as a GA) presents the additional problem of modelling crossover. Given two parent sets, the distribution of child sets can be defined in terms of the child distribution for a pair of parents, averaged over all possible pairs from the parent set. While this would not be too difficult for an exact model of the search space, any form of state aggregation complicates the issue somewhat. A simple crossover model for barrier tree states, based on the average Hamming distance between points from each parent set, was explored in Hallam (2006), but unfortunately produced poor predictions of the actual crossover probabilities. Other, more sophisticated approximations are certainly possible, although the nonlinearity typical of crossover models is likely to make analysis more difficult (particularly the calculation of a gradient for parameter optimisation).

8.2 Alternative Schedule Parameterisation

In §6.4 we touched upon the idea of adjusting schedule parameters according to some variable other than time. Many statistics are available for use as input to such a parameter generator, with examples including cost, cost difference of most recent transition, last decision (accept/reject), time since last uphill (or downhill, rejected, accepted) step; some of these may also be measured over more than one iteration (averaging over some number of iterations, or an exponentially-weighted average over the whole history). Since arbitrarily complex combination of these measures are possible, optimising their influence on the search parameters is likely to be difficult, and very sensitive to any predetermined form chosen for the relationship.

Most of these cannot be incorporated into a model without either destroying the Markov property, or expanding the state vector to include history information. The exception is cost, which can be computed for any state in isolation. In fact, if cost is the *only* statistic controlling the search parameters, the model is simplified greatly by the time-independence of the transition matrices. This was exploited in §7.4, where we optimised FPT over all such transition matrices.

The chief advantage of directly linking parameters to the state of the search is that behaviour may be prompted according to the actual performance, rather than being based solely on a prediction of the heuristic's likely positions. This prospect is particularly attractive in such situations as ours, where a model tends to consistently over-estimate progress.

8.3 Schedule Extrapolation

The ability to construct a parameter schedule optimised for a specific problem is of little practical use if the optimisation process requires enumeration of the entire search space, as is usually the case with the barrier tree model. Ideally, one would like to be able to construct a good schedule from either a limited sampling of the search space, or full analysis of a small sample of problems.

By "limited sampling" of the search space, we mean a sampling so brief that the associated cost might be outweighed by the benefit from optimised parameters. Modelling navigation across small regions of the search space may provide hints as to the size, depth and frequency of local optima. However, the applicability of this information to the entire problem requires careful consideration, as locally observed properties of the cost landscape may give a misleading impression of the whole.

Where there is a need to solve a large number of similar problems, it may be feasible to produce a barrier tree model of a few representative problem instances, on which the search parameters may be optimised (§5.7). Search space models need not be built through exhaustive search; our model of the hurdle problem (§4.1) is an example of a model built directly from the problem specification. By their very nature, hard problems do not have cost landscapes with structure so transparently visible as for the hurdle problem. Nonetheless, for particular classes of problems there may be shortcuts allowing the construction or estimation of a barrier tree model (or some similar abstraction) without the need for exhaustive enumeration of all search points.

Appendix A

Miscellaneous Proofs

A.1 The Fixed Point for Constant-Temperature Annealing is a Boltzmann Distribution

We begin with the observation that our neighbour matrix m satisfies a *detailed balance* constraint; if $\mathbf{s}$ is a vector of the sizes of the states, then

$$\forall i, j \quad m_{ij} s_j = m_{ji} s_i.$$

This applies to both barrier tree transition probabilities, and the mutation probabilities for the underlying search space. Given any undirected graph, transition probabilities produced from normalised edge weights will satisfy this criterion.

For a random walk (i.e. with m as the transition matrix), the fixed point is given by $\mathbf{s}$, as

$$\begin{array}{rrcl} & \forall i \ \ \sum_j m_{ij} s_j & = & \sum_j m_{ji} s_i. \\ \implies & \forall i \ \ (m\mathbf{s})_i & = & s_i \\ \implies & m\mathbf{s} & = & \mathbf{s} \end{array} .$$

This is exactly what we would expect: for a random walk, the equilibrium distribution weights each state according to size.

We now apply the same process for the transition matrix for simulated annealing. Assume $i \neq j$ (when $i = j$, the detailed balance criterion is satisfied trivially); then $w_{ij} = m_{ij} A_{ij}$. We can describe A_{ij} in terms of A_{ji}, as

$$\begin{aligned} A_{ij} &= \begin{cases} e^{-\beta(c_i - c_j)} & c_i > c_j \\ 1 & c_i \leq c_j \end{cases} \\ e^{\beta(c_i - c_j)} A_{ij} &= \begin{cases} 1 & c_i > c_j \\ e^{-\beta(c_j - c_i)} & c_i \leq c_j \end{cases} \\ e^{\beta(c_i - c_j)} A_{ij} &= A_{ji}, \end{aligned}$$

giving us $A_{ij}e^{-\beta c_j} = A_{ji}e^{-\beta c_i}$. It is then trivial to verify that $w_{ij}s_je^{-\beta c_j} = w_{ji}s_ie^{-\beta c_i}$; through the same calculation as for the random walk, we have the vector $\mathbf{s} \otimes e^{-\beta \mathbf{c}}$ as a fixed point for w.

A.2 An Upper Bound for the Number of Minima on a Cost Landscape with Hypercubic Connectivity

If S is a connected search space, with points and neighbourhood connectivity isomorphic to the vertices and edges of the L-cube (with $L > 0$), then 2^{L-1} is an upper bound for the number of local minima in S.

The search space may be divided equally in two by making a cut through the centre of the L-cube, parallel to one pair of opposite faces; equivalently, the set of all possible bit-strings of length L may be divided according to the value at one arbitrarily selected bit-position. Consider pairs of opposite points (bit-strings identical save for the selected bit-position); each point is the opposite to exactly one other point (to which it is also connected), and the relationship is symmetric. The entire search space must then be the disjoint union of 2^{L-1} such pairs, each of which can contain no more than one minimum (if the costs are equal, then both points belong to the same minimum). Therefore a landscape with connectivity graph isomorphic to the L-cube can have no more than 2^{L-1} minima.

A.3 Number of possible populations over a finite search space

Given a search space of size S, the number of possible (distinct) populations of size P is given by $\binom{S+P-1}{P}$.

To demonstrate that this is so, we introduce an alternative representation for populations. Choosing any arbitrary ordering of the points in the search space, such that an agent may traverse the space from "start" to "finish", passing through every state. Suppose that such an agent may place any number of population members at each step, so long as the total placed equals P. The behaviour of the agent may be described as a list of the symbols $\{\text{move}, \text{place}\}$, indicating whether the agent should next move onward or place a population member. The list must therefore contain exactly P "place" symbols, and (asserting that the agent must finish traversing the space even after the whole population has been placed) exactly S-1 "move" symbols.

The inverse mapping (from populations to lists) may be defined through a reversal of the algorithm. Suppose now that the agent traverses a populated space, writing a "move" symbol for every step in the walk, and writing a "place" symbol for every population member encountered.

Since the inverse is well-defined, the mapping between instruction lists of this form and populations must therefore be bijective, with the number of possible populations equal to the number of possible instruction lists, i.e. the binomial coefficient $\binom{S+P-1}{P}$.

A.4 A Guarantee of Convergence is not a Guarantee that FPT Exists

The average FPT, given by $\sum_{t=0}^{\infty} t\mathbb{P}(T_{fp} = t)$ is not well-defined for every algorithm, as the infinite sum could potentially diverge. Perhaps the most notable example of this is the FPT for descent, which is convergent iff the search space contains no local minima. If there is at least one minimum other than the global optimum, then there is a nonzero probability that descent will visit it (averaging over all possible starting points and all possible random neighbour selections). If the minimum is visited, the search will remain there indefinitely, and thus the FPT from that point will be infinite. Regardless of how small the minimum may be, the contribution from this possibility will dominate the FPT calculation for the search. In fact, we can clearly see that a necessary (but not sufficient) condition for the average FPT to converge is that the algorithm itself is capable of converging to the optimum from any state

This criterion does not however guarantee that the average FPT will converge to a finite value; it is possible to construct a combination of algorithm and problem instance such that the average FPT diverges.

Let $S = \{s_1, s_2\}$ be a search space, with $n(s_1) = \{s_2\}$, $n(s_2) = \{s_1\}$, and $c(s_1) > c(s_2)$. Consider the following (extremely contrived) algorithm:

Algorithm 5: An "exponentially slow" algorithm

```
state ← $s_1$;
$t \leftarrow 1$;
while termination criterion not satisfied do
    next ← state if $t > 1$ and $t$ is a power of two then
        if A uniformly random number from $[0, 1)$ is less than $\frac{1}{2}$ then
            state ← random-neighbour(state);
        end
    end
    if $c(next) \leq c(state)$ then
        state ← next;
    end
    $t \leftarrow t + 1$;
end
```

If the algorithm is in state s_1 at time t, then the probability of moving to the optimum is $\mathbb{P}(s_1 \to s_2, t) = \frac{1}{2}\left[\log_2 t \in \mathbb{Z}^+\right]$, i.e. a half if t is a power of two, otherwise zero. Let $\mathbb{P}(T_{fp} = t)$ denote the probability that the first transition to the optimum occurs at time t. Clearly this is zero when t is not a power of two. We have:

$$\begin{aligned}
\mathbb{P}\left(T_{fp} = 2^k\right) &= \mathbb{P}\left(s_1 \to s_2, 2^k\right) \prod_{i=0}^{2^k-1} \left(1 - \mathbb{P}\left(s_1 \to s_2, 2^i\right)\right) \\
&= \frac{1}{2} \prod_{i=0}^{2^k-1} \left(1 - \frac{1}{2}\left[\log_2 i \in \mathbb{Z}^+\right]\right) \\
&= \frac{1}{2} \prod_{i=1}^{k-1} \frac{1}{2} \\
&= 2^{-k}.
\end{aligned}$$

The non-power-of-two terms in the product equal unity, and thus may be discarded, producing a product over powers of two. We shall employ a similar trick for summation over the probabilities. The probability of reaching the global optimum before time T approaches 1 as $T \to \infty$, as

$$\begin{aligned}
\lim_{T\to\infty} \sum_{t=1}^{T} T_{fp} &= \lim_{T\to\infty} \sum_{t=1}^{T} t^{-1}\left[\log_2 t \in \mathbb{Z}^+\right] \\
&= \lim_{T\to\infty} \sum_{i=1}^{\lfloor \log_2 T \rfloor} 2^{-i} \\
&= \lim_{T\to\infty} 1 - 2^{-\lfloor \log_2 T \rfloor} \\
&= 1.
\end{aligned}$$

The average FPT however is divergent, as

$$\begin{aligned}
\langle T_{fp} \rangle &= \lim_{T\to\infty} \sum_{t=1}^{T} t T_{fp} \\
&= \lim_{T\to\infty} \sum_{t=1}^{T} t t^{-1}\left[\log_2 t \in \mathbb{Z}^+\right] \\
&= \lim_{T\to\infty} \sum_{t=1}^{T} t\left[\log_2 t \in \mathbb{Z}^+\right] \\
&= \lim_{T\to\infty} \sum_{i=1}^{\lfloor \log_2 T \rfloor} 1 \\
&= \lim_{T\to\infty} \lfloor \log_2 T \rfloor
\end{aligned}$$

does not exist. Hence we have evidence that, perhaps counterintuitively, it is possible for an algorithm to be guaranteed to converge in the infinite-time limit, whilst having a divergent FPT.

A.5 FPT on Direct Search Space Models

If a heuristic is to operate directly on the search space, then structure or symmetries of that space can often be used to compute FPT without the slightly cumbersome procedure in §7.3.2. For bitstring search spaces with Hamming-neighbourhood connectivity, we can easily produce expressions for the FPT of several particularly simple special-case algorithms. The average FPT for random sampling is trivially given by $2^{L-1} - \frac{1}{2}$; note that for consistency with our other results, we do not count the starting point as a cost evaluation.

A.5.1 Optimal Walk

The decision function is intuitively obvious: a neighbour should be accepted if and only if it is closer (as measured by Hamming distance) to the global optimum. For a point at distance k from the optimum, there will be k acceptable transitions. The probability, on each iteration, of an acceptable neighbour being selected is k/L, and therefore the average time before a transition is L/k. The average FPT from a point at distance k from the optimum is therefore $\sum_{i=1}^{k} L/i$; averaging over the entire search space gives a binomially weighted sum of these FPTs:

$$\langle T_{fp} \rangle = 2^{-L} \sum_{k=0}^{L} \binom{L}{k} L \sum_{i=1}^{k} 1/i.$$

A.5.2 Random Walk

Let the average FPT from a point at distance k from the optimum be denoted by T_k. Then the FPTs are related by a series of equations:

$$T_k = \begin{cases} 0 & k = 0 \\ 1 + T_{L-1} & k = L \\ 1 + \frac{k}{L} T_{L-1} + \left(1 - \frac{k}{L}\right) T_{L+1} & 0 < k < L \end{cases}$$

Writing these constraints in matrix form gives $\mathrm{t} = M\mathrm{t} + \mathbf{1}$, where M is of the form

$$M = \frac{1}{L} \begin{bmatrix} 0 & L-1 & & & \\ 2 & 0 & L-2 & & \\ & \ddots & & \ddots & \\ & & L-1 & 0 & 1 \\ & & & L & 0 \end{bmatrix}$$

L	Random sampling	Optimal walk	Random walk
0	0.0	0.000000	0.000000
1	0.5	0.500000	0.500000
2	1.5	1.750000	2.500000
3	3.5	3.500000	7.250000
4	7.5	5.604167	17.166667
5	15.5	7.973958	36.958333
10	511.5	22.359034	1171.894841
15	16383.5	39.376254	35598.583146
20	524287.5	58.091850	1111387.767063

TABLE A.1: Average FPT for several simple algorithms on search spaces for bitstrings of various length.

and $\mathbf{t}$ is a column vector of $T_1 \ldots T_L$, which can then be found by simply evaluating $(I - M)^{-1}\mathbf{1}$. The average FPT is given by a binomial sum over $T_0 \ldots T_k$.

Note that for $L > 2$, a random walk is more than twice as slow as random sampling (table A.1); this is why memoryless searches such as Simulated Annealing sometimes struggle to escape plateau regions.

Appendix B

Defective Matrices

A matrix is said to be *defective* if it has at least one eigenvalue λ_d whose *algebraic* and *geometric* multiplicity do not match (Lanczos, 1957; Golub and Loan, 1983; Horn and Johnson, 1985). The former is the multiplicity with which the factor $(\lambda - \lambda_d)$ occurs in the characteristic polynomial of the matrix, the latter is the dimension of the associated eigenspace, and is therefore equivalently the maximum number of linearly independent eigenvectors one can find with eigenvalue λ_d.

Common operations implemented through eigenvalue decomposition may fail if naïvely applied to defective matrices; this is demonstrated to occur in the popular numerical computation environments "MatLab" and "Octave", both of which can produce incorrect results, often without warning.

B.1 Examples

The simplest possible defective matrix is probably the matrix

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

which has characteristic equation $(1-\lambda)^2 = 0$, and therefore has 1 as a repeated eigenvalue. However, $(A - I)\,\mathbf{x} = \mathbf{0}$ has a solution space of dimension 1; all eigenvectors of A are multiples of $[\,1\ 0\,]^\top$.

A slightly less trivial example is provided by any matrix of the form

$$\begin{bmatrix} 1 & (1-a) & & & \\ & a & (1-a) & & \\ & & a & \ddots & \\ & & & \ddots & (1-a) \\ & & & & a \end{bmatrix}$$

(with all omitted entries being zero). Any such matrix is triangular, and therefore has eigenvalues equal to the entries on the main diagonal. Thus for an $n \times n$ matrix, 1 appears once as an eigenvalue, and a appears $n-1$ times. When we attempt to find the eigenspace associated with a, we solve $(M - aI)\mathbf{x} = 0$. Of course, the diagonal of $(M - aI)$ disappears (except for the '1'), giving us constraints of the form $(1-a)x_i = 0$ for $i = 3 \ldots n$, and $(1-a)x_1 + (1-a)x_2 = 0$. All x_i for $i > 2$ are therefore zero, with the remaining two entries summing to zero; thus all eigenvectors for a are of the form $[\alpha, -\alpha, 0, \ldots, 0]^{\mathrm{T}}$, and the eigenspace has dimension 1.

B.1.1 A Non-Triangular Example

Matrices of the form described above are nonsingular and stochastic (for $0 < a \leq 1$), and could therefore represent the state transitions for some process, although the obvious structure would likely not go unnoticed. A matrix need not be triangular (or capable of being permuted to a triangular matrix through reassignment of row and column indices) in order to be defective. Listing B.1 shows a MatLab session exploring a (stochastic, nonsingular) defective matrix which merely contains a triangular sub-matrix.

```
>> M

M =

    0.4352    0.0425    0.1000    0.0500    0.1709    0.0093
    0.0815    0.0098    0.1000    0.0500    0.2495    0.4535
         0         0    0.7000    0.1000         0         0
         0         0         0    0.7000         0         0
    0.3144    0.3270    0.0500    0.0500    0.3349    0.2830
    0.1688    0.6207    0.0500    0.0500    0.2446    0.2542

>> det(M)

ans =

   -0.0029

>> sum(M)

ans =

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

```
>> [evecs, evals] = eig(M)

evecs =

   -0.2245   -0.8041   -0.3396    0.0162    0.0658   -0.0658
   -0.4542    0.3540   -0.0952   -0.7466   -0.2257    0.2257
         0         0         0         0    0.8427   -0.8427
         0         0         0         0         0    0.0000
   -0.5949   -0.0267    0.8423    0.0689   -0.3151    0.3151
   -0.6240    0.4768   -0.4075    0.6615   -0.3677    0.3677


evals =

    1.0000         0         0         0         0         0
         0    0.4166         0         0         0         0
         0         0    0.0343         0         0         0
         0         0         0   -0.4168         0         0
         0         0         0         0    0.7000         0
         0         0         0         0         0    0.7000

>> (M^.5)^2 - M

ans =

  -0.0000   -0.0000   -0.0000   -0.0263 + 0.0005i   -0.0000   -0.0000
  -0.0000   -0.0000   -0.0000   -0.0211 + 0.0110i   -0.0000   -0.0000
        0         0   -0.0000   -0.1000                   0         0
        0         0         0   -0.0000                   0         0
  -0.0000   -0.0000   -0.0000   -0.0842 + 0.0024i   -0.0000   -0.0000
  -0.0000   -0.0000   -0.0000    0.0357 + 0.0013i   -0.0000   -0.0000
```

Listing B.1: MatLab session demonstrating incorrect behaviour for a calculation involving a defective matrix.

The eigenvalue 0.7 is repeated, but has only one eigenvector, and so the MatLab function "`eig`" does not return a full set of linearly independent eigenvectors: observe that the fifth is the negation of the sixth.

B.1.2 An Ergodic Example

The defective matrix

$$D = \begin{bmatrix} 0.3129 & 0.3348 & 0.1848 \\ 0.4098 & 0.3973 & 0.4942 \\ 0.2772 & 0.2679 & 0.3210 \end{bmatrix}$$

is stochastic (the columns sum to 1) and is clearly ergodic, as all entries are strictly positive. Aside from 1, the only eigenvalue is 2^{-6}, which is repeated but has only a single eigenvector. D was constructed by arbitrarily choosing by assuming a non-trivial

Jordan normal form (Golub and Loan, 1983; Horn and Johnson, 1985)[1], with Jordan block matrix J, and then finding (largely by trial and error) a matrix S and an integer k such that SJ^kS^{-1} was stochastic and ergodic. In this case, D was given by $D = SJ^6S^{-1}$, where

$$S = \begin{bmatrix} 2 & -8 & 0 \\ 3 & 6 & -10 \\ 2 & 2 & 10 \end{bmatrix}$$

and

$$J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}.$$

B.2 Impact on Calculations

Any computation requiring the eigenvalue decomposition of a matrix is only well-defined so long as that decomposition exists, which in the case of defective matrices is not so. Raising a matrix to a noninteger power is an example of one such operation, which if attempted within MatLab will sometimes produce an incorrect result with no warning at all. In Listing B.1, we see that even the simple identity $M = \left(M^{1/a}\right)^a$ will sometimes fail to hold, with the nonzero entries of the difference being comparable to the entries in M. In fact, this also happens for the trivial defective matrix A described earlier, with *all* noninteger powers of that matrix producing the identity.

In the examples given above, MatLab produced correct results when raising the ergodic matrix to a noninteger power, but produced large inaccuracies when this test was repeated with the other matrices. For *some* defective matrices, MatLab will warn that a computation involved a singular (or near-singular) matrix (the matrix of eigenvectors is singular for a defective matrix), but this warning was not given for any of our examples. One possibility is that inaccuracies occur because the matrix of eigenvectors is, due to imperfect machine representation, computed to be *near*-singular (and ill-conditioned), a condition which may be difficult to detect. It is nonetheless still a little unsettling that the absence of a warning is not sufficient to ensure the validity of the result.

Defective matrices can be considered to be something of a rarity, as having a repeated eigenvalue is obviously a prerequisite for defectiveness. If the eigenvalues of a matrix are chosen randomly, the probability of a value being repeated is only nonzero when one takes machine representation into account. However, the distribution of eigenvalues over the set of "matrices likely to occur in calculations" is very different; if one is restricted to a particular class of random matrices, then the probability of defectiveness may become

[1]All matrices can be reduced to a form $A^{-1}JA$ where J is a matrix of Jordan blocks. For non-defective matrices, J is a diagonal matrix.

significant. In summary, the output of a process is very unlikely to be defective unless there is some mechanism causing it to be so.

In this thesis, there is just a single occasion where an operation sensitive to this problem is used: in §3.1.3 a matrix exponential is used to compute the effect of applying a Poisson-distributed number of mutations. A selection of matrices involved were tested and verified to be non-defective. The acceptance matrix in §7.3 is defective[2], but eigenvalue-dependent operations are not performed (and would likely be of very little significance).

In the case of the ergodic matrix described earlier, a small error was present in each of the eigenvalues found by MatLab, causing them to be regarded as inequal. As a result, the diagonalisation represented a (very close) approximation to the actual matrix. Although unintentional, this small perturbation prevented the matrix of eigenvectors from being singular (or very close to singular), providing a greater accuracy in the final result. This effect cannot be relied upon, but does at least seem to suggest that the conditions under which a calculation may become corrupted are very strict.

[2]Although defective in the form specified, the elements along the main diagonal may be chosen arbitrarily (as there is no difference in our model between rejection and acceptance of a self-transition), and thus the eigenvalues may chosen at will.

Appendix C

Supplementary Data

C.1 Barrier Tree Statistics

This section provides some statistics for the barrier tree structures of each of the test problems. All models were produced from level-accessible barrier trees, with the exception of the Max-SAT 'LCS' problem, for which the states were instead taken to be the level-connected sets.

For the binary perceptron and spin-glass problems, the number of variables (also $\log_2$ of the number of points in the search space) is the number preceding the first hyphen in the problem name. The Max-SAT problems were all 20-variable problems, with the exception of '40v', which was a 40-variable problem.

For each problem instance, statistics are (where appropriate) provided separately for the (local or global) minimum states ('M') and for all states (i.e. including the non-minimum states) ('A'). These statistics are the number of states, the number of distinct costs possessed by those states, and the average, minimum and maximum costs of those states. Additionally, the 'propMinima' column shows the proportion of the search space which is occupied by minima.

Only integer costs are possible for our problems (excluding the hurdle problem, not listed here). Spin-glass problems with an odd number of variables are a special case. In such a problem, inverting a single bit-position affects the correlations with all other states, of which there will be an even number. The sum over an even number of elements in $\{-1, +1\}$ will always produce an even number, so a single-bit mutation (and by extension, any series of such mutations) will always adjust the total cost by a multiple of two. For this reason, all of the spin-glass problems with an odd number of variables (i.e. all but the first three) have the property that all possible costs have identical residue modulo 2.

Name	States		prop Min	Dist costs		Average Cost		Min C	Max Cost	
	A	M		A	M	A	M		A	M
1	36	8	1.907e-05	28	3	14.125	2.950	2.0	29.0	4.0
10	43	15	3.147e-05	28	6	14.375	4.667	2.0	29.0	7.0
11	40	11	2.098e-05	29	6	14.000	3.909	1.0	29.0	6.0
12	48	16	3.338e-05	32	5	14.750	5.086	3.0	34.0	7.0
13	40	10	2.480e-05	29	3	14.750	2.962	2.0	30.0	4.0
14	38	8	2.956e-05	30	2	14.250	2.613	2.0	31.0	3.0
15	37	8	1.240e-05	29	3	15.750	3.462	2.0	30.0	4.0
16	35	9	2.098e-05	26	3	14.125	3.000	2.0	27.0	4.0
17	42	14	2.956e-05	29	5	15.375	4.710	3.0	31.0	8.0
18	34	5	1.049e-05	28	2	15.125	2.455	2.0	29.0	3.0
19	38	8	1.621e-05	30	3	15.250	2.471	2.0	31.0	5.0
40v	85	36	4.575e-10	50	5	23.375	2.628	1.0	50.0	5.0
LCS	92	7	5.245e-05	32	3	15.750	3.545	3.0	34.0	5.0
orig	38	7	5.245e-05	32	3	15.750	3.545	3.0	34.0	5.0

FIGURE C.1: Max-SAT: barrier tree statistics

Name	States		prop Min	Dist costs		Average Cost		Min C	Max Cost	
	A	M		A	M	A	M		A	M
10-1	25	2	1.953e-03	20	1	22.500	12.000	12.0	31.0	12.0
10-2	29	6	5.859e-03	19	2	22.500	13.667	13.0	31.0	14.0
10-3	33	12	1.172e-02	20	4	22.500	15.333	14.0	33.0	17.0
15-1	23	4	6.714e-04	19	2	52.500	36.545	36.0	72.0	38.0
15-2	24	4	1.831e-04	20	1	52.500	35.000	35.0	73.0	35.0
15-3	19	2	3.052e-04	18	1	52.500	35.000	35.0	69.0	35.0
17-1	31	6	4.578e-05	24	2	68.000	46.333	45.0	91.0	47.0
17-2	31	4	7.629e-05	25	2	68.000	45.200	44.0	92.0	46.0
17-3	29	8	1.984e-04	22	2	68.000	48.154	48.0	90.0	50.0

FIGURE C.2: Spin glass: barrier tree statistics

Name	States		prop Min	Dist costs		Average Cost		Min C	Max Cost	
	A	M		A	M	A	M		A	M
10-12-1	36	27	2.637e-02	10	5	7.477	5.111	2.0	11.0	9.0
10-12-2	106	99	9.863e-02	9	5	7.477	5.594	3.0	11.0	7.0
10-12-3	72	65	6.738e-02	9	6	7.477	5.000	3.0	11.0	8.0
12-11-1	134	124	3.027e-02	12	9	6.741	3.323	0.0	11.0	8.0
12-11-2	115	106	2.856e-02	10	5	6.741	3.829	2.0	11.0	6.0
12-11-3	26	16	3.906e-03	11	7	6.741	3.500	1.0	11.0	7.0
13-10-1	220	212	3.247e-02	9	6	5.000	2.635	1.0	9.0	6.0
13-10-2	319	309	3.894e-02	11	7	5.000	2.301	0.0	10.0	6.0
13-10-3	112	103	1.575e-02	9	5	5.000	2.403	1.0	9.0	5.0
14-11-1	211	202	1.379e-02	10	6	6.652	4.142	2.0	11.0	7.0
14-11-3	464	453	2.789e-02	12	9	6.652	4.031	0.0	11.0	8.0

FIGURE C.3: Binary perceptron: barrier tree statistics

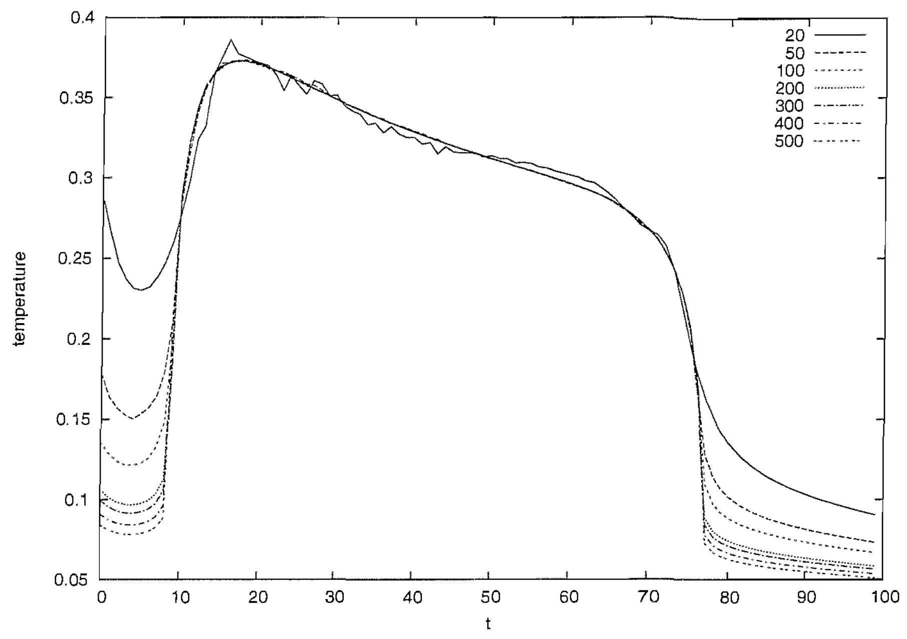

FIGURE C.4: Snapshots of a WYA Max-SAT schedule at various iterations of the SCG optimisation.

C.2 Optimal Annealing Schedules

C.2.1 Convergence

It was found that by the 500th iteration, the SCG algorithm had not converged entirely, but produced very small improvements in the low-temperature regions at the beginning and end of the schedules. This is most likely due to the choice of parameterisation; for the temperature to reach zero, the inverse temperature must become infinite. Where the temperature appears to take increasingly small steps, the value seen by the SCG algorithm may be growing constantly or even divergent, confusing the optimisation process.

Figures C.4 and C.5 illustrate this with a series of "snapshots" of a schedule after 20, 50, and then every 100 iterations of the SCG optimisation. Excluding the initial and final low-temperature regions, the shape of the final schedule is visible, albeit with some random noise, after only 20 iterations; after 50 iterations the noise has almost disappeared, and the central section of the schedule is already indistinguishable from that of the final (500th iteration) schedule.

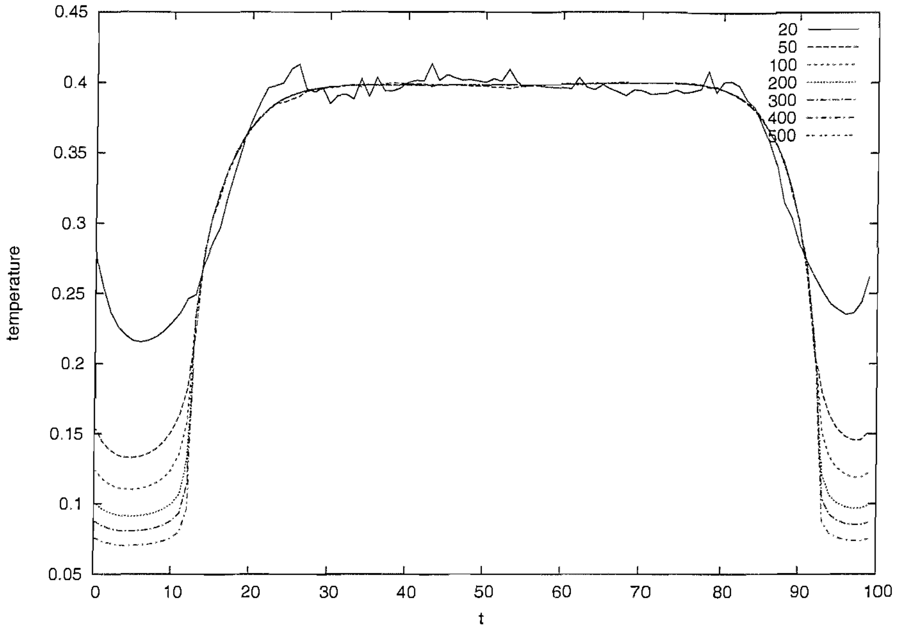

FIGURE C.5: Snapshots of a BSF Max-SAT schedule at various iterations of the SCG optimisation.

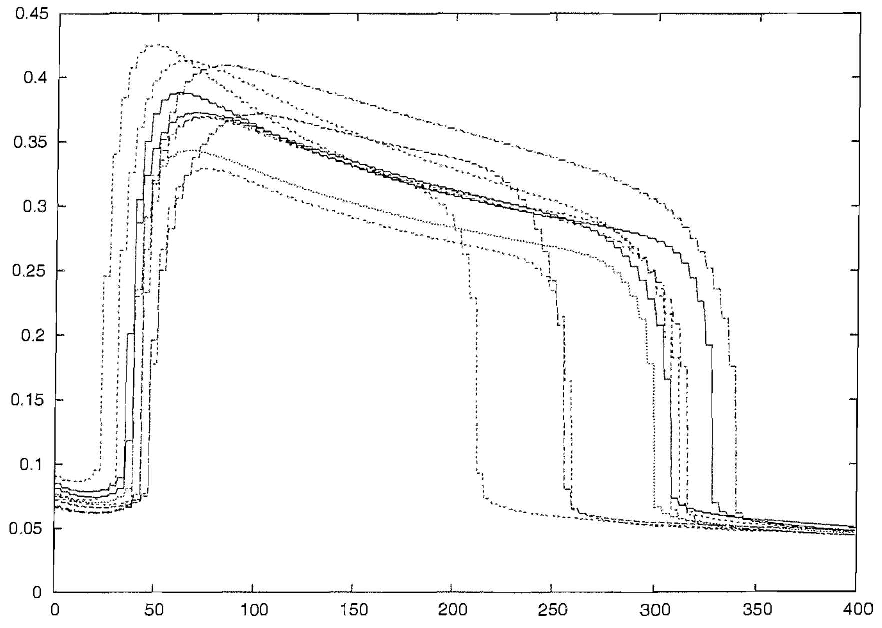

FIGURE C.6: Superimposed WYA schedules (length 400) for a series of Max-SAT problems

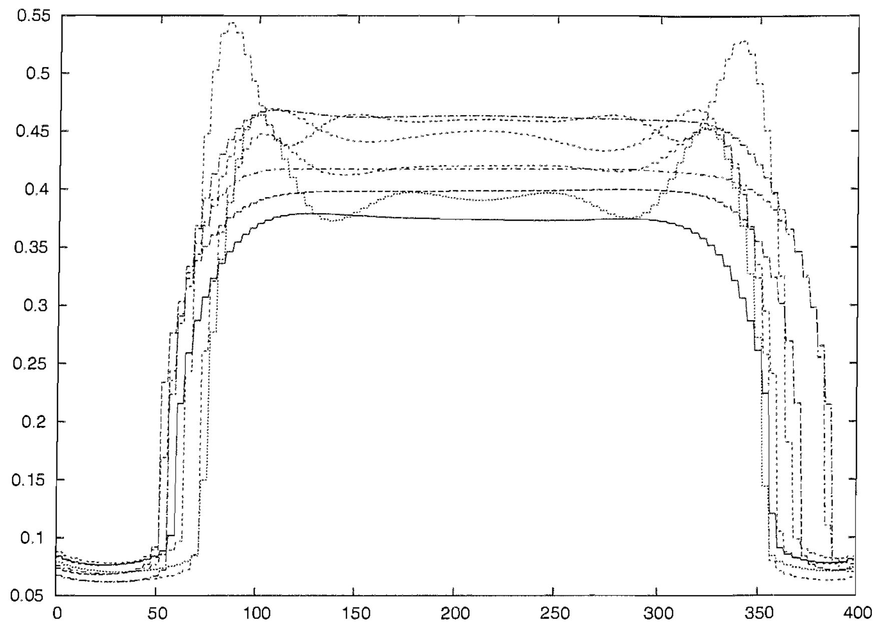

FIGURE C.7: Superimposed BSF schedules (length 400) for a series of Max-SAT problems

C.2.2 Max-SAT Schedules

WYA-optimal schedules for all of the Max-SAT problems shared a common shape (fig. C.6), although the maximum temperature and the time at which the final cooling stage was entered varied considerably. This is also approximately true of most of the BSF schedules (fig. C.7), although several also showed some oscillation in temperature, with varying amplitude.

Amongst the BSF-optimised Max-SAT schedules was one significant exception to the rule, featuring three prominent peaks; this is shown and discussed in §5.5.

C.2.3 Binary Perceptron Schedules

The WYA perceptron schedules (fig. C.8) resemble the WYA Max-SAT schedules, but with the initial low-temperature region replaced by one of oscillating temperature. In three of the graphs shown, this oscillation is trivial, with a single spike preceding the rest of the schedule. Two graphs show a series of oscillations (eight, ten cycles), with amplitude decreasing toward the height of the remainder of the schedule.

Each of the BSF perceptron schedules shown in fig. C.9 is worthy of comment:

Top left: A repeated four-stage cycle containing alternate high, narrow spikes and lower, wider peaks, each delimited by a brief region of very low temperature.

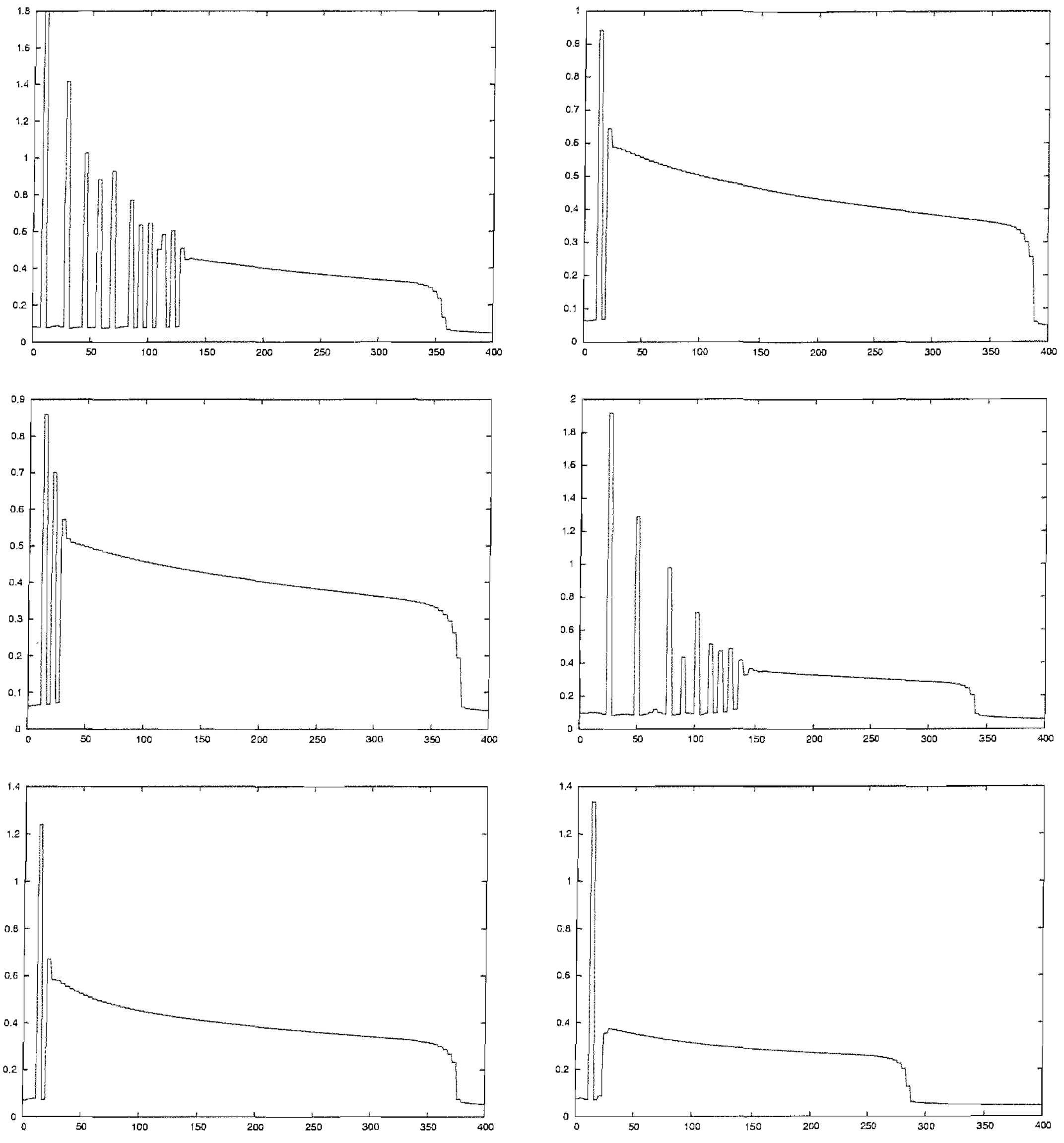

FIGURE C.8: WYA-optimal schedules for binary perceptron problems in 10 (left) and 12 (right) variables.

Centre left: What may appear at first glance to be random noise is actually fairly structured. The schedule alternates between very high and very low temperatures, occasionally visiting an intermediate temperature during a transition. From every low-temperature point, a high temperature is reached within at most two steps.

Bottom left: This schedule has unfortunately failed to converge after 500 iterations of SCG.

Top right: Unusually, this schedule has a only a central section of oscillating temperature; it is possible that this may not be the final form of the schedule (i.e. that further optimisation is possible).

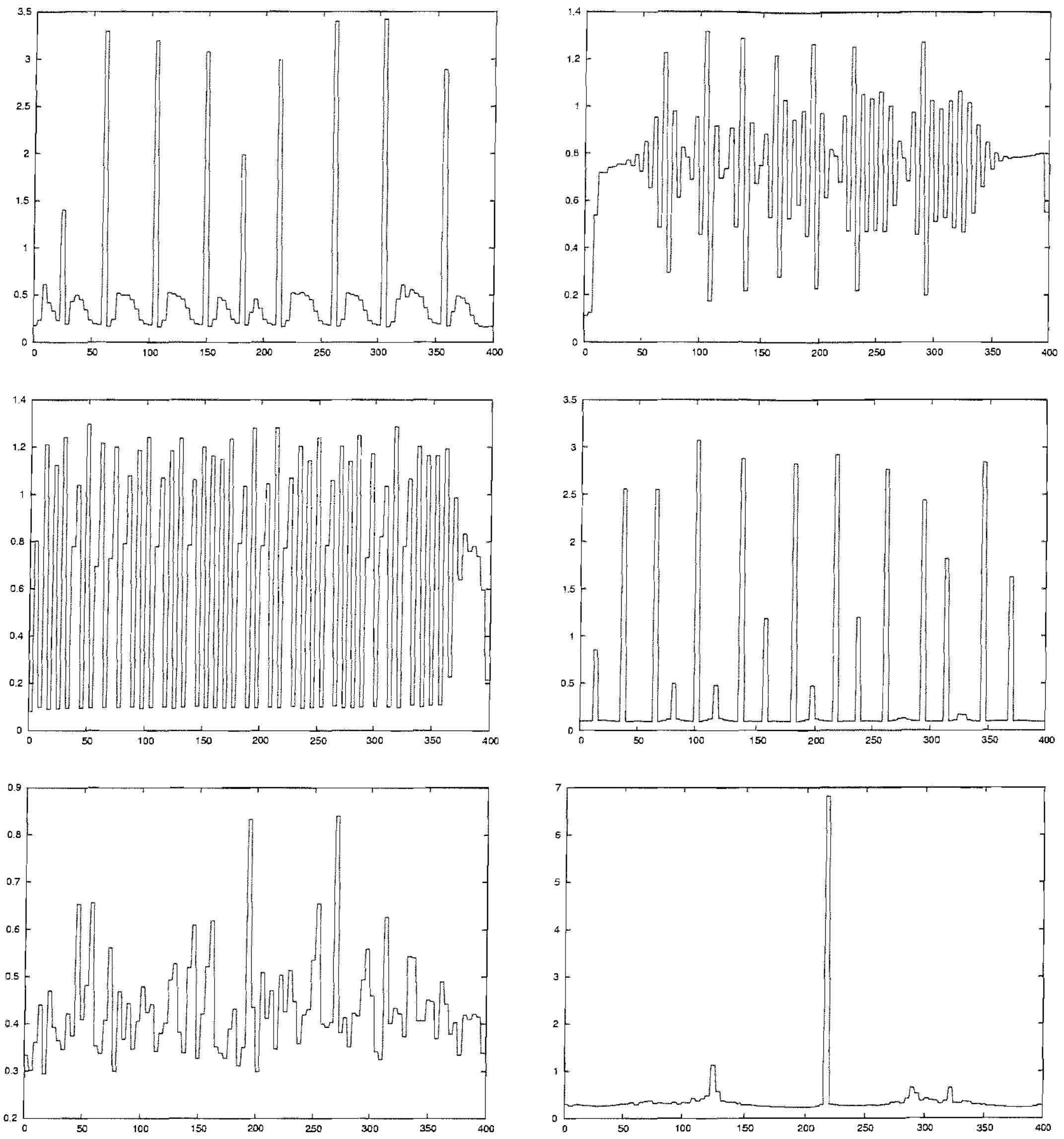

FIGURE C.9: BSF-optimal schedules for binary perceptron problems in 10 (left) and 12 (right) variables.

Centre right: A series of high spikes approximately 30 — −50 iterations apart, between which some smaller spikes are emerging. The combination of spike sizes and to some extent the spacing between them may be an artifact of the choice of schedule parameterisation, which makes it difficult for the optimiser to "shift" features with respect to time. Given a partially-optimised schedule with a set of pre-existing spikes, adjusting the period requires many temperature values to be manipulated gradually, if less efficient intermediate schedules are to be avoided. Inserting smaller spikes between the larger ones is an alternative requiring fewer modifications, and with less deceptive gradient information.

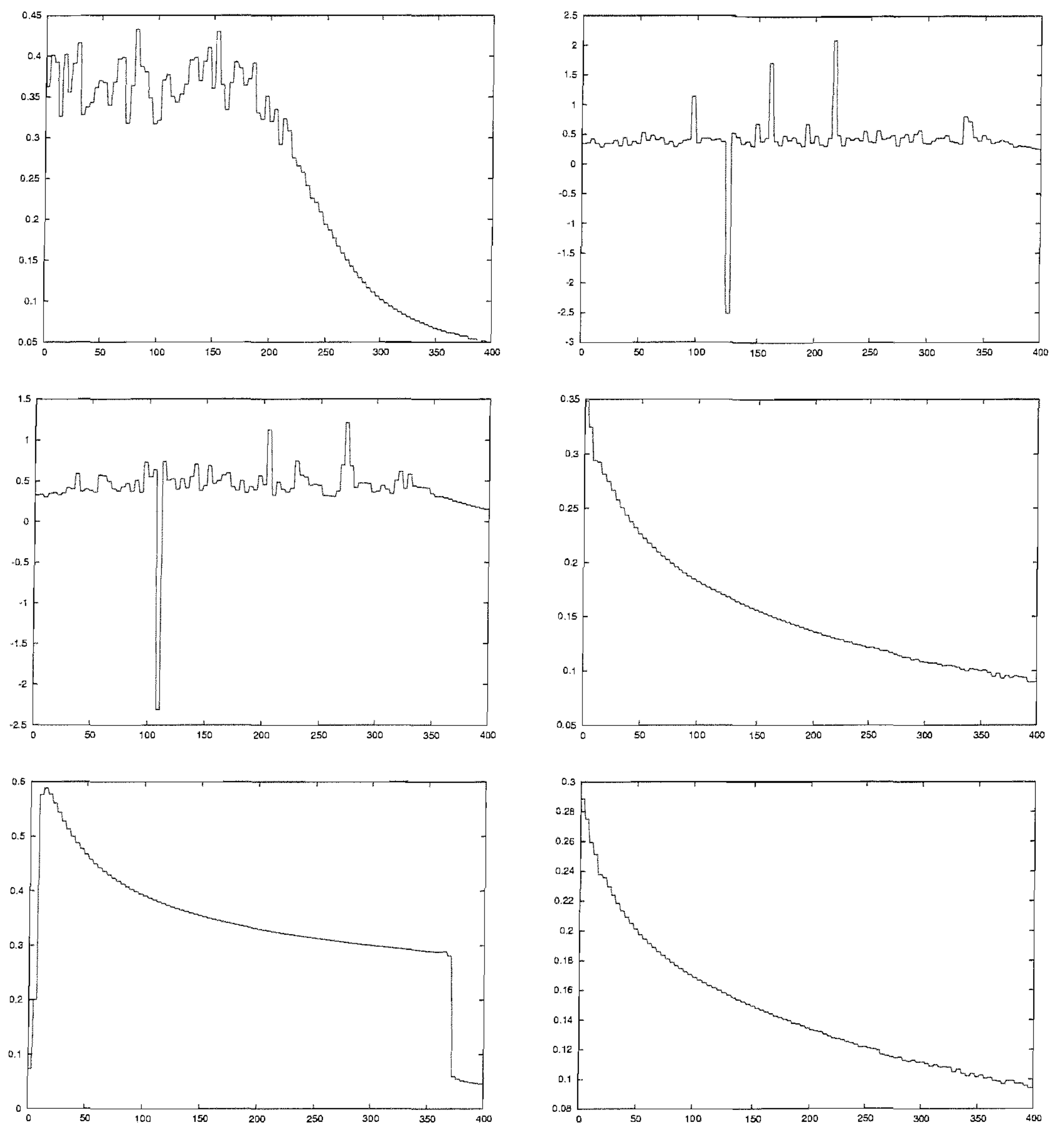

FIGURE C.10: WYA-optimal schedules for spin-glass problems in 10 (left) and 15 (right) variables.

Bottom right: It is difficult to tell whether this schedule is "genuine" or is the result of a divergent optimisation process.

C.2.4 Spin-Glass Schedules

Of the WYA spin-glass schedules (fig. C.10), only three converged fully. Two of these are monotone-decreasing, one (lower left) has the same form as the Max-SAT schedules.

Two of the BSF schedules (fig. C.11) have a similar 'U' shape to that seen for BSF hurdle problem schedules in §5.2; the reasons for this are unclear, as there is no obvious similarity between the hurdle and spin-glass problems.

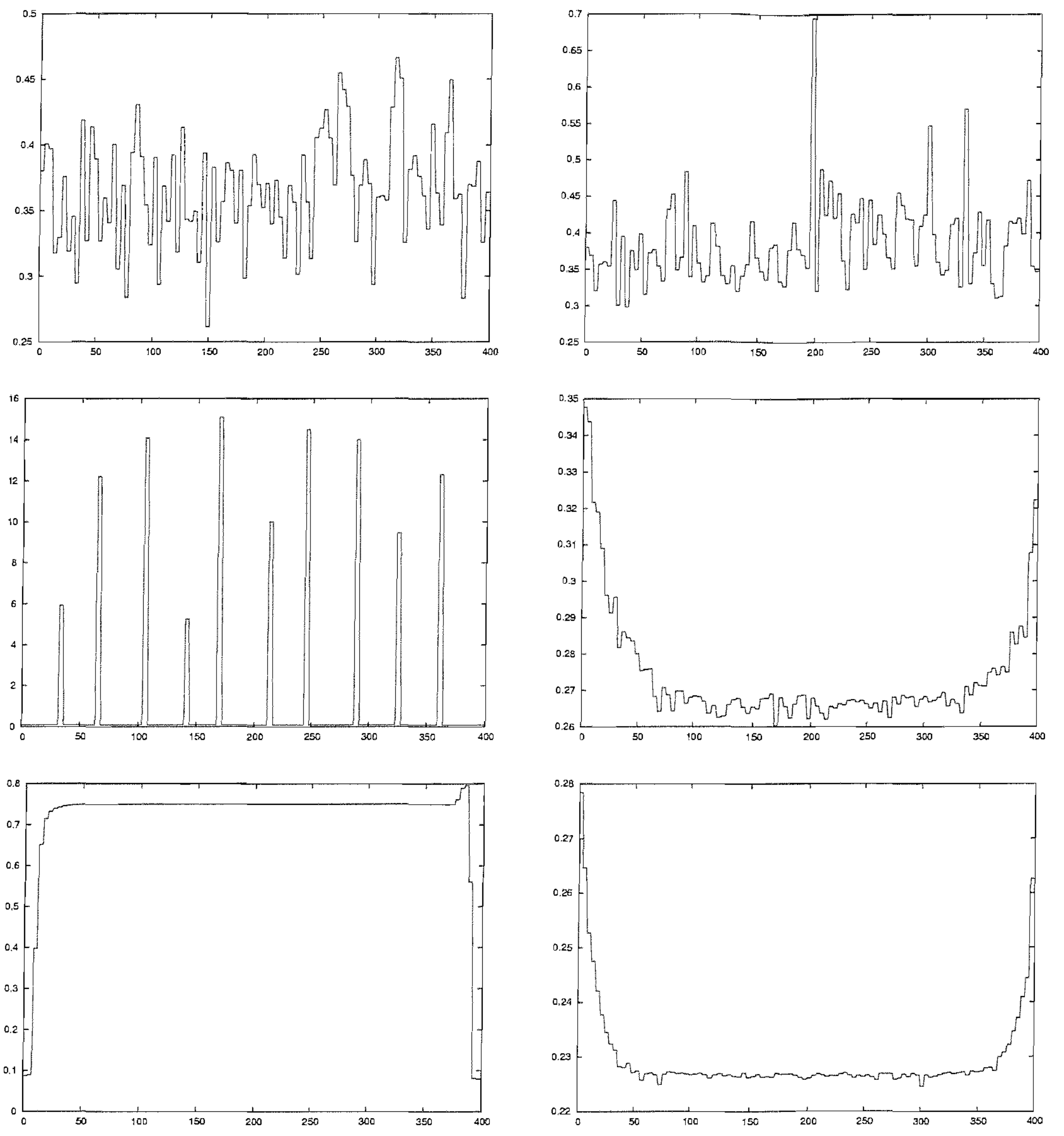

FIGURE C.11: BSF-optimal schedules for spin-glass problems in 10 (left) and 15 (right) variables.

C.2.5 Schedule Performance

Figures C.12 and C.13 show the predicted and actual performance of each optimised schedule. To aid comparison, each schedule was evaluated for both WYA and BSF cost, regardless of which the schedule was optimised for. The "predicted" costs are those output by the Markov model; the "observed" costs were obtained experimentally by passing the schedule to a simulated annealing algorithm running on the appropriate test problem; the observed costs were averaged over 100 000 runs.

On a few problems, the SCG algorithm repeatedly diverged, consistently producing schedules containing (negative temperatures. In these cases, the transition matrices

Name	Pred	Obs WYA	Obs BSF
hurdle16	1.182	1.179640e+00	1.017270e+00
hurdle64	7.8659	7.865425e+00	7.534280e+00
sat10	2.0788	2.868930e+00	2.838230e+00
sat11	1.2956	2.099270e+00	2.052590e+00
sat12	3.1077	3.821610e+00	3.793990e+00
sat13	2.0537	2.471860e+00	2.427450e+00
sat14	2.0706	2.356970e+00	2.328830e+00
sat15	2.2947	2.915820e+00	2.831150e+00
sat16	2.1172	2.781220e+00	2.741950e+00
sat17	3.1243	3.875160e+00	3.848270e+00
sat18	2.038	2.362180e+00	2.338840e+00
sat19	2.0012	2.048130e+00	2.041160e+00
sat1	2.0291	2.331700e+00	2.302990e+00
satorig	3.0582	3.361760e+00	3.338550e+00
per10-12-1	2.0115	2.030550e+00	2.015340e+00
per10-12-2	4.13	4.310290e+00	4.136370e+00
per10-12-3	3.6339	3.727340e+00	3.569900e+00
per12-11-1	0.79338	9.171900e-01	7.107000e-01
per12-11-2	2.1018	2.150240e+00	2.075470e+00
per12-11-3	1.0449	1.105370e+00	1.063060e+00
spin10-1	12	1.200000e+01	1.200000e+01
spin10-2	-66.641	1.332244e+01	1.311715e+01
spin10-3	14.172	1.428116e+01	1.421167e+01
spin15-1	36.046	3.629676e+01	3.610986e+01
spin15-2	35	3.503212e+01	3.503208e+01
spin15-3	35	3.508466e+01	3.508464e+01
spin17-1	45.188	4.575586e+01	4.553988e+01
spin17-2	44.454	4.472656e+01	4.451492e+01
spin17-3	48.008	4.811398e+01	4.802330e+01

FIGURE C.12: Performance of WYA-optimised schedules.

were no longer stochastic, and the predicted costs were divergent. These results may be identified easily in figs. C.12 and C.13; they are included for completeness only and should not be compared to the others.

C.3 Short Schedules

Particularly short WYA annealing schedules were sometimes (particularly on Max-SAT problems) found to have a different form (fig. C.14 vs. fig. C.6).

In §5.4.4 it was noted that, regardless of length, the optimised schedules for a particular Max-SAT problem had initial and final low-temperature regions of approximately equal length. As the schedule length is reduced, the width of the central higher-temperature

Name	Pred	Obs WYA	Obs BSF
hurdle16	0.8607	2.224110e+00	8.531700e-01
hurdle64	7.3506	8.676795e+00	7.328350e+00
sat10	2.016	2.946990e+00	2.733690e+00
sat11	-2.0104e+30	2.744940e+00	2.052840e+00
sat12	3.041	3.851480e+00	3.707040e+00
sat13	2.0067	2.599710e+00	2.355550e+00
sat14	2.009	2.489990e+00	2.263970e+00
sat15	2.123	3.133550e+00	2.730100e+00
sat16	2.0227	2.954480e+00	2.634680e+00
sat17	3.0299	3.927800e+00	3.740370e+00
sat18	2.0055	2.450040e+00	2.273000e+00
sat19	-2.9067e+21	2.204590e+00	2.030210e+00
sat1	2.0038	2.450800e+00	2.257060e+00
satorig	3.0028	3.494410e+00	3.242300e+00
per10-12-1	2	2.441930e+00	2.000270e+00
per10-12-2	3.6867	5.279460e+00	3.846240e+00
per10-12-3	3.3746	3.905360e+00	3.478690e+00
per12-11-1	0.32747	2.211320e+00	4.202700e-01
per12-11-2	2.0004	2.676760e+00	2.006640e+00
per12-11-3	1.0023	1.458960e+00	1.028790e+00
per13-10-1	1	1.524460e+00	1.000860e+00
per13-10-2	0.80929	1.162720e+00	8.537600e-01
per13-10-3	1.0002	1.522470e+00	1.009390e+00
spin10-1	12	1.218840e+01	1.200000e+01
spin10-2	13.001	1.357022e+01	1.300626e+01
spin10-3	14.004	1.478525e+01	1.402481e+01
spin15-1	36.14	3.636950e+01	3.633886e+01
spin15-2	35	3.505144e+01	3.503370e+01
spin15-3	35	3.508482e+01	3.508236e+01
spin17-1	-13071	4.647802e+01	4.549686e+01
spin17-2	-1.0061e+80	4.482928e+01	4.445966e+01
spin17-3	46.137	4.834804e+01	4.800984e+01

FIGURE C.13: Performance of BSF-optimised schedules.

region reduces also. The non-existence of this region for sufficiently short schedules is consistent with this behaviour.

Of six spin-glass schedules produced (fig. C.16), three showed similar behaviour to the short Max-SAT schedules; one diverged during optimisation, and the remaining two produced "normal" Max-SAT-like schedules. The binary perceptron schedules (fig. C.15) either diverged during optimisation (in three cases), or produced normal Max-SAT-like schedules.

From §C.1 we see that the binary perceptron and spin-glass problems had smaller search spaces than the Max-SAT problems (10-15 variables vs. 20 variables), but did not have significantly fewer states, and had minima covering a greater proportion of the search

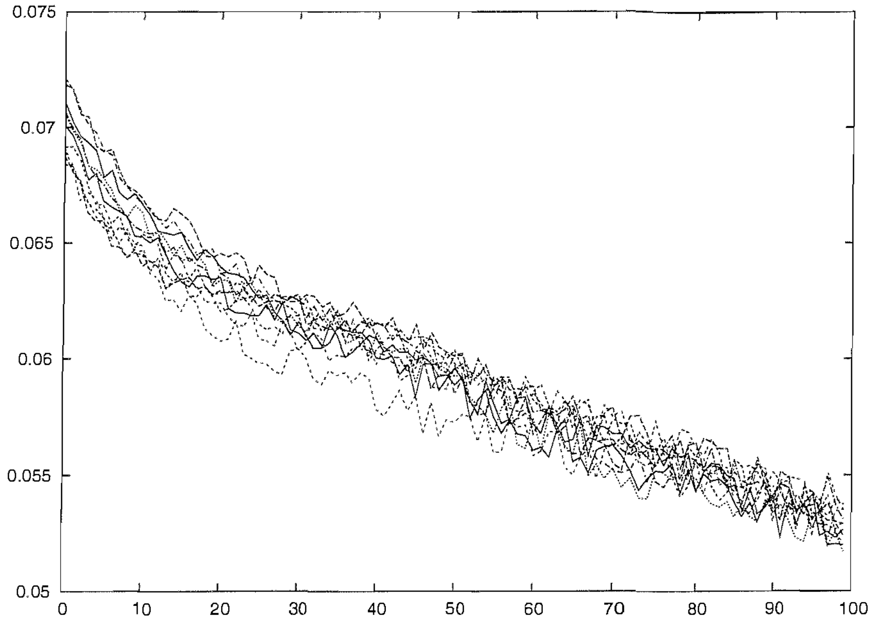

FIGURE C.14: Superimposed WYA schedules (length 100) for a series of Max-SAT problems

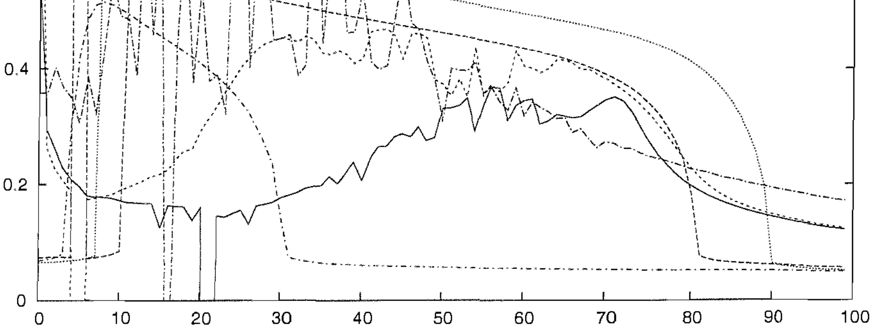

FIGURE C.15: Superimposed WYA schedules (length 100) for a series of binary perceptron problems

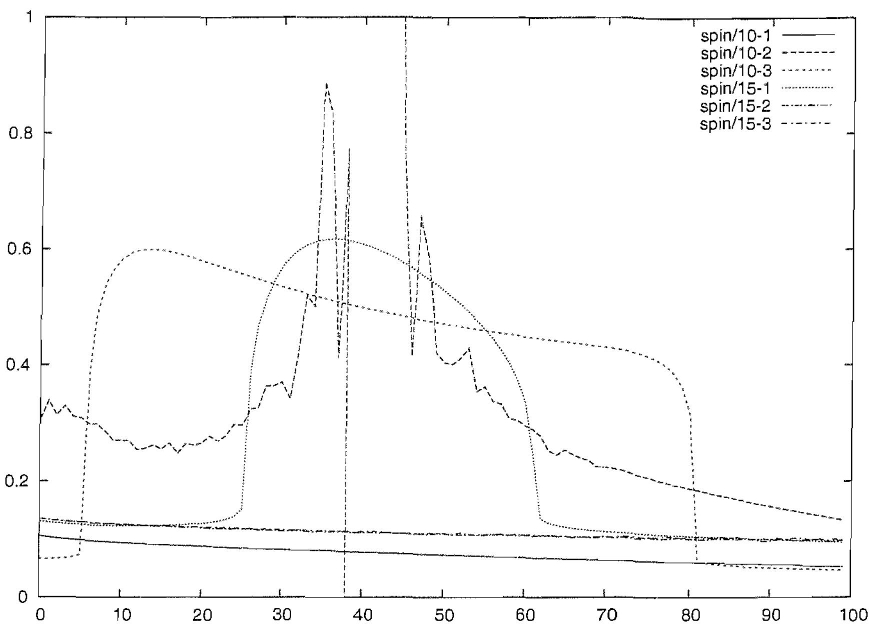

FIGURE C.16: Superimposed WYA schedules (length 100) for a series of spin-glass problems

space. This suggests that state transitions, including those to or from local minima, might occur less frequently when traversing Max-SAT problems, or might be clustered in one region of the space. The schedules in fig. C.14 could resemble descent simply because the probability of reaching any local minima within 100 iterations is very small.

C.4 Constrained Periodic Schedules

Schedules with a series of evenly-spaced square peaks and troughs (fig. C.17) were studied. The shape of the schedule is determined by four parameters; the maximum and minimum temperature, the peak width and the trough width.

For each problem, BSF cost was optimised over these parameters, with a schedule length of 400. The two widths were represented by real-valued parameters, with first iteration of each peak or trough being interpolated according to the fractional part of the previous width. For example, a peak width of 1.5 would be interpreted as a peak of width 1, with the first value of the following trough being half way between the peak and trough heights. Strictly speaking, this allows our imposed schedule form to be broken, as the optimisation process is able to exploit this leeway. It is necessary however as a practical measure to provide the optimiser with some hints as to which direction to follow; applying a floor function would produce a series of featureless plateaus. An alternative solution would be to find an optimiser designed to be applied to a combination of continuous and discrete parameters.

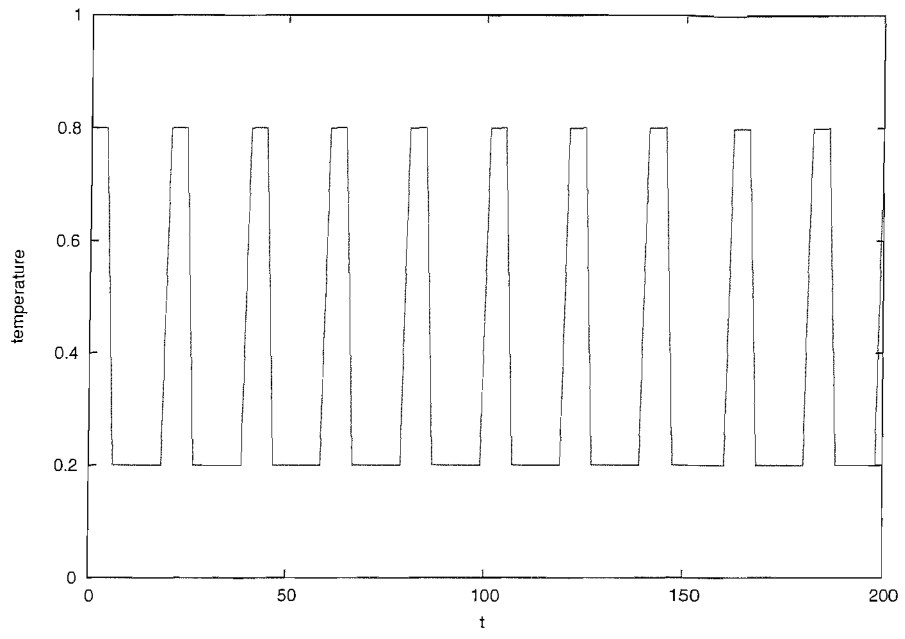

FIGURE C.17: A "square wave" periodic schedule with maximum temperature 0.8, minimum 0.2, peak width 6 and trough width 14.

The results (fig. C.18) are quite consistent in having very narrow peaks; for all but two problems, the peak width was between 1 and 2. Note that the peak width was artificially constrained to prevent it being reduced below 1; this was accomplished by reporting a very high cost for those schedules. Again, this was to prevent the creation of plateaus in the parameter space (this time for peak widths less than zero). Note that the effect of zero-width peaks (i.e. a flat schedule) could still be obtained by equating the peak and trough temperatures; there are several cases where this appears to have occurred, including both of the exceptions to the peak width rule described above.

The trough widths are comparatively wide, varying from around ten iterations to a hundred. Two spin-glass runs had troughs which were very large, but both peak and trough temperatures were very small; this is a flat schedule as described above, with temperature low enough to effectively be a descent.

Comparing the freely optimised and constrained periodic schedules, we see a substantial difference between performance on the binary perceptron and Max-SAT problems. On the latter, the freely optimised schedules have better performance in every case, as we would expect. The binary perceptron results however show the constrained periodic schedules performing better than the free schedules.

One possible explanation is that the very narrow spikes seen in the periodic schedules are crucial to performance. The "free" schedule is unable produce these spikes due to the effect of coarse-graining; the block size for these results was four iterations. The heights of the temperature peaks in both cases are consistent with this theory, with the narrow

Name	Free cost	Per cost	loLen	hiLen	lo	hi
hurdle16	0.8607	1.731	14	1	0.643045	3.30404
hurdle64	7.3506	8.256	25	1	0.474766	6.85918
per10-12-1	2	2	14.048	1.0029	0.0515013	6.33473
per10-12-2	3.6867	3.6201	8	1.0704	0.149533	454.36
per10-12-3	3.3746	3.1171	7	1.4306	0.172167	6.56556
per12-11-1	0.32747	0.25734	12.988	1	0.0577868	2.04344e+11
per12-11-2	2.0004	2.0001	13	1	0.0902201	7.32332
per12-11-3	1.0023	1.0019	25	1	0.139548	3.23593
per13-10-1	1	1	11.592	1.0454	0.0414921	6.53467
sat1	2.0038	2.0041	100	1.6645	0.350508	0.575043
sat10	2.016	2.0168	105.84	1.4562	0.382629	0.551329
sat11	-2.0104e+30	1.2167	81	1.4566	0.361755	0.726058
sat12	3.041	3.0434	57	1	0.289461	1.11564
sat13	2.0067	2.0077	69	1.0351	0.414233	0.964413
sat14	2.009	2.0098	57	1.6309	0.359932	0.949758
sat15	2.123	2.1294	99	1.176	0.441638	0.510569
sat16	2.0227	2.0245	58	1.736	0.383715	1.00465
sat17	3.0299	3.0336	40	1.4959	0.345985	1.50013
sat18	2.0055	2.0067	39	1.6025	0.295186	1.31146
sat19	-2.9067e+21	2.0001	93.996	1.2724	0.339443	0.29215
satorig	3.0028	3.0031	34	2.7888	0.445831	0.473059
spin10-1	12	12	52.493	1.1174	0.0725374	0.889521
spin10-2	13.001	13.001	16	1.9816	0.0590772	10.8147
spin10-3	14.004	14.004	18.837	1.0034	0.729661	2.91494
spin15-1	36.14	36	13.517	1.1659	0.0817394	5.34474
spin15-2	35	35	326.86	1.5289	0.0371789	0.083022
spin15-3	35	35	283.53	11.526	0.0321626	0.0813273

FIGURE C.18: Results of BSF optimisation with "square wave" form, temperature peaks adjustable and included in schedule length.

spikes of the constrained periodic schedules being higher than the four-iteration-wide peaks of the free schedules.

Bibliography

P. W. Anderson. What statistical mechanics has to say to computer scientists. *Physica A: Statistical and Theoretical Physics*, 140:405–409, 1986.

W. Benfold, J. Hallam, and A. Prügel-Bennett. Optimal annealing schedules for larger problems. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1119–1126, Sept. 2005a. ISBN 0-7803-9363-5.

W. Benfold, J. Hallam, and A. Prügel-Bennett. Optimal parameters for search using a barrier tree markov model. Submitted to Theoretical Computer Science, 2005b.

P. Blaudeck and K. H. Hoffmann. Ground states for condensed amorphous systems: Optimizing annealing schemes. *Computer Physics Communications*, pages 293–299, Feb. 2003.

K. Boese and A. Kahng. Best-so-far vs. where-you-are: Implications for optimal finite-time annealing. *Systems and Control Letters*, 22(1):71–8, 1994.

A. Bölte and U. W. Thonemann. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, 92:401–416, 1996.

T. Boseniuk and W. Ebeling. Boltzmann-, Darwin- and Haeckel-strategies in optimization problems. *Lecture Notes in Computer Science: Parallel Problem Solving from Nature*, 496:430–444, 1991.

T. Boseniuk, W. Ebeling, and A. Engel. Boltzmann and Darwin strategies in complex optimization. *Physics Letters A*, 125:307–310, 1987.

A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: an algorithm for satisfiability. *Random Structures and Algorithms*, 27:201–226, 2005.

V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

M. Christoph and K. H. Hoffmann. Scaling behaviour of optimal simulated annealing schedules. *J. Phys. A: Math. Gen.*, 26:3267–3277, 1993.

H. Cohn and M. Fielding. Simulated annealing: Searching for an optimal temperature schedule. *SIAM Journal on Optimization*, 9:779–802, April 1999.

S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, May 1971.

P.-J. Courtois. On time and space decomposition of complex structures. *Communications of the ACM*, 28(6):590–603, June 1985.

P.-J. Courtois and P. Semal. Computable bounds for conditional steady-state probabilities in large markov chains and queueing models. *IEEE Journal on Selected Areas in Communications*, 4(6):926–937, Sept. 1986.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

B. Derrida, R.B. Griffiths, and A. Prügel-Bennett. Finite-size effects and bounds for perceptron models. *Journal of Physics A*, 24:4907–4936, 1991.

S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1–2):51–82, 2002.

G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimzation algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

S. F. Edwards and P. W. Anderson. Theory of spin glasses. *Journal of Physics F*, 5: 965–974, 1975.

C. Flamm, I. L. Hofacker, P. F. Stadler, and M. T. Wolfinger. Barrier trees of degenerate landscapes. *Z. Phys. Chem.*, 216:155–173, 2002.

A. Franz and K. H. Hoffmann. Optimal annealing schedules for a modified tsallis statistics. *Journal of Computational Physics*, 176:196–204, 2002.

M. R. Garey and D. S. Johnson. *A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0716710447.

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

G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 1983. ISBN 0-946536-05-8.

D. R. Greening. Parallel simulated annealing techniques. *Physica D*, 42:293–306, 1990.

B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

J. Hallam. *Barrier Trees for Studying Search Landscapes.* PhD thesis, School of Electronics and Computer Science, University of Southampton, Feb. 2006.

J. Hallam and A. Prügel-Bennett. Barrier trees for search analysis. In *GECCO*, pages 1586–1587, 2003.

J. Hallam and A. Prügel-Bennett. Crossover and barrier based markov models. In *Proceedings of CEC 2005*, volume 2, pages 1661–1666, 2005a.

J. Hallam and A. Prügel-Bennett. Large barrier trees for studying search. *IEEE Transactions on Evolutionary Computation*, 9(4):385–397, 2005b.

P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303, Dec. 1990.

K. H. Hoffmann and P. Salamon. The optimal simulated annealing schedule for a simple model. *J. Phys. A: Math. Gen.*, 23:3511–3523, 1990.

R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985. ISBN 0-521-38632-2.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

W. Krauth and M. Mézard. Storage capacity of memory networks with binary couplings. *Journal de Physique*, 50:3057–3066, 1989.

C. Lanczos. *Applied Analysis.* Sir Isaac Pitman & Sons, Ltd., 1957.

F. T. Lin and C. Y. Kao. Incorporating genetic algorithms into simulated annealing. In *Proceedings of the Fourth International Symposium on Artificial Intelligence*, pages 290–297, 1991.

S. W. Mahfoud and D. E. Goldberg. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing*, 21:1–28, Jan. 1995.

N. Metropolis, A. Rosenbluth, M.Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6): 1087–1092, 1953.

M. Mézard, G. Parisi, and M. Virasoro. *Spin Glass Theory and Beyond.* World Scientific, Nov. 1987. ISBN 9971-50-115-5.

M. Miki, T. Hiroyasu, T. Yoshida, and T. Fushima. Parallel simulated annealing with adaptive temperature determined by genetic algorithm. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 257–261, 2002.

D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, July 1992.

R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, July 1999.

I. T. Nabney. *Netlab: Algorithms for Pattern Recognition.* Advances in Pattern Recognition. Springer, 2001. ISBN 1-85233-440-1.

J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

G. Parisi. Magnetic properties of spin glasses in a new mean field theory. *J. Phys. A*, 13:1887–1895, 1980.

L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

A. Prügel-Bennett. Symmetry breaking in population-based optimization. *IEEE Transations on Evolutionary Computation*, 8(1):63–79, Feb. 2004a.

A. Prügel-Bennett. When a genetic algorithm outperforms hill-climbing. *Theoretical Computer Science*, 320(1):135–153, 2004b.

D. Janaki Ram, T. H. Sreenivas, and K. Ganapathy Subramaniam. Parallel simulated annealing algorithms. *Journal of Parallel and Distributed Computing*, 37:207–212, Sept. 1996.

D. Sherrington and S. Kirkpatrick. Solvable model of a spin glass. *Physics Review E*, 35:1792–1796, 1975.

W. M. Spears and K. A. De Jong. Analyzing GAs using markov models with semantically ordered and lumped states. In Richard K. Belew and Michael D. Vose, editors, *Foundations of Genetic Algorithms 4*, pages 85–100. Morgan Kaufmann, San Francisco, CA, 1997.

P. N. Strenski and S. Kirkpatrick. Analysis of finite length annealing schedules. *Algorithmica*, 6:346–366, 1991.

R. P. White and H. R. Mayne. Optimal annealing schedules for two-, three-, and four-level systems using a genetic algorithm approach. *Journal of Chemical Physics*, 112 (18):7964–7978, July 2000.

David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.