Optimal Parameters for Search Using a Barrier Tree Markov Model

W. Benfold, J. Hallam, A. Prügel-Bennett *

School of Electronics and Computer Science,
University of Southampton, SO17 1BJ, UK

Abstract

The performance, on a given problem, of search heuristics such as simulated annealing and descent with variable mutation can be described as a function of, and optimised over, the parameters of the heuristic (e.g. the annealing or mutation schedule). We describe heuristics as Markov processes; the search for optimal parameters is then rendered feasible by the use of level-accessible barrier trees for state amalgamation. Results are presented for schedules minimising "where-you-are" and "best-so-far" cost, over binary perceptron, spin-glass and Max-SAT problems. We also compute first-passage time for several "toy heuristics", including constant-temperature annealing and fixed-rate mutation search.

Key words: Optimal search, optimal annealing schedule, variable mutation, barrier tree

1 Introduction

Efficient solution of real-world problems is dependent not only on the selection of an appropriate search heuristic, but also on the choice of appropriate parameters for that algorithm. For any particular combination of heuristic, search space, and measure of efficiency, there exists a set of parameters for which the heuristic performs optimally, insofar as it is deemed to be at least as efficient as any other parameter set. Optimal parameters have been found analytically for problems of either very small size [1,2], or problems with a

* Corresponding author
Email addresses: wb02r@zepler.org (W. Benfold), jbh02r@ecs.soton.ac.uk (J. Hallam), apb@ecs.soton.ac.uk (A. Prügel-Bennett).

high degree of symmetry [3,4]. In both cases however, the practical application as a means of tuning parameters is rather limited, as the parameter values are calculated based on the specification of the cost function, from which the global minimum can be determined trivially. A reasonably complex problem is considered in [5], where schedules are produced to guarantee convergence of the distribution of states to those of optimum cost, in the infinite time limit. We shall concern ourselves only with finite schedules.

It is our belief that discovery of the optimal parameters for a heuristic operating on one problem of a given type may provide some insight into the choice of parameters for other problems of the same type. We hope 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. Further, if the problems have some similarity of structure, we could expect that heuristic behaviour, and thus the best choice of heuristic parameters, might be similar across those problems.

We model a search algorithm as a Markov process (§2), where each transition represents one iteration of the algorithm. The state vector for the process is a probability distribution for the states of the search algorithm. An analytic expression for the average cost (§3) for the search can be computed from this Markov model, this expression can then be minimised over the parameter space for the search algorithm.

We choose to restrict ourselves to search algorithms 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 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 $O(2^{LP})$.

In §4.1 we examine a toy problem with a highly symmetric state space; this symmetry is exploited to produce a reduced model of the problem 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.

In §4.2, we look at some instances of "real" problems (Max-SAT, binary perceptron, spin glass); these are traditional "hard problems", which do not possess such obvious structural symmetry, and thus cannot be reduced so trivially. 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 com-

pute. We combat this by amalgamating points of the search space to produce a model of the problem with a relatively small number of states (§5).

An equivalence relation is defined over the points of the search space to produce a "barrier tree" of equivalence classes corresponding to local minima and saddle points of the cost function, which are used as the states of our Markov model. This is, however, an approximation to the original problem, insofar as the transition probabilities are calculated by an unweighted average over the search points belonging to a state, thus assuming that all such points are equally likely to be visited.

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, 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 §6. 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 §7 we present results for an annealing schedule optimised over a set of Max-SAT problems.

Finally, in §8, 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 Time (§8.1) with those of single-parameter versions of simulated annealing and mutation search (§8.4).

2 Markov Models of Search Heuristics

As discussed earlier (§1), we shall only consider search heuristics for which the state space for the heuristic is the same as the search space; the state vector for the Markov model will be a probability distribution for the search space. The initial distribution, which we will refer to as $\mathbf{p}_0$ shall be assumed to represent a random distribution of starting positions chosen from the search space.

(In §5, we consider a reduced model of a search space, for which the elements

of $\mathbf{p}_0$ correspond to large regions of the search space, and are thus proportional in value to the size of those regions.)

2.1 General

For both simulated annealing (§2.2) and descent with variable mutation (§2.3), the behaviour at a given iteration depends on the number of iterations performed previously (which we shall refer to as "time"). Thus the transition matrices used to represent the search will themselves be time-dependent; we shall refer to them as $w(t)$.

In both cases however, the time dependency is indirect, via a function $\beta(t)$, known as the "schedule". This function is only evaluated for "iteration numbers" which are positive integers up to a maximum number T, also known as the "schedule length".

In §3, we will find the "cost" of a heuristic, which we shall write as a function of β. We later optimise the heuristic by finding the set of parameters which produces the smallest cost; since we use a gradient-based method to do so, we will describe here not only the matrices $w(t)$, but also their derivatives with respect to the values of the schedule $\beta(t)$.

For both of the following heuristics, a definition of a neighbourhood around a point in the search space is required. A good neighbourhood definition is essential for efficient search, but cannot be constructed from the cost function alone. It is normal for the neighbourhood function to incorporate representation details from the problem, and thus provide a structure which would otherwise be invisible to the heuristic. We assume that such a definition is provided in the form of a mutation matrix m where the term m_{ji} represents the probability of a random mutation from states $i \to j$. The presence of neighbours with differing probabilities is typically the result of state amalgamation (§5.2).

2.2 Simulated Annealing

Simulated annealing[6] is a popular heuristic for optimisation problems, inspired by the physical process of annealing, whereby a metal is heated and then slowly cooled to reach a state of lower energy. In our case, the "energy" is cost, and the "temperature" is used to compute the likelihood of accepting an uphill transition.

With each iteration, one point from the neighbourhood of the current position

is evaluated. If the costs of the current position and the neighbour are given by c_i and c_j respectively then the probability of adopting the chosen neighbour as the new "current position" is given by the corresponding element of the *acceptance matrix* A, defined to be

$$A_{ji} = \begin{cases} 1 & c_i - c_j \geq 0 \\ e^{\beta(t)(c_i - c_j)} & c_i - c_j \leq 0 \end{cases}$$

where t is the "time" (iteration number). Here, the parameter $\beta(t)$ (known as the *inverse temperature*) controls the probability of accepting an uphill step, which approaches 1 as $\beta(t) \to 0$, and decreases to 0 for large β.

Transition probabilities

The transition matrix $w(t)$ depends only on $\beta(t)$. Let m be a (problem-specific) matrix such that m_{ji} is the probability of a single mutation causing a transition from state i to state j. Each non-diagonal entry of $w(t)$ is the probability of a mutation occurring and being accepted; the diagonal entries represent self-transitions *and* rejected mutations. The elements of the matrix $w(t)$ are therefore given by

$$\begin{aligned} w(t)_{ji} &= \mathbb{P}(i \to j) \\ &= \mathbb{P}((i \to j) \text{ accepted}) + \delta_{ji}\mathbb{P}((i \to \textit{anything}) \text{ rejected}) \\ &= A_{ji}m_{ji} + \delta_{ji}\textstyle\sum_k (1 - A_{ki})\, m_{ki} \\ &= A_{ji}m_{ji} + \delta_{ji}\left(\textstyle\sum_k m_{ki} - \sum_k A_{ki}m_{ki}\right) \\ &= (A \otimes m)_{ji} + \delta_{ji}\left(\mathbf{1}^\top - \mathbf{1}^\top (A \otimes m)\right)_i \end{aligned}$$

where $\otimes$ denotes the operation of *element-wise* matrix multiplication, i.e. $(A \otimes B)_{ji} \equiv A_{ji}B_{ji}$, and $\mathbf{1}$ is a (column) vector of 1s. We define the operations "sum" and "diag" as follows:

$$\begin{aligned} \text{sum}(M) &= \mathbf{1}^\top M \\ (\text{diag}(\mathbf{v}))_{ij} &= \delta_{ij} v_i \end{aligned}$$

allowing us to write $w(t)$ in the slightly more intuitive form

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

from which it should be clear that $w(t)$ is stochastic - the terms added to the diagonal of $(A \otimes m)$ are the values by which the column sums are deficient.

Calculation of Gradient

The first derivatives of the transition matrices are given by:

$$\frac{\partial w(t)}{\partial \beta(t)} = \frac{\partial \left(A(t)\right)}{\partial \beta(t)} \otimes m - \operatorname{diag}\left(\operatorname{sum}\left(\frac{\partial \left(A(t)\right)}{\partial \beta(t)} \otimes m\right)\right)$$

and

$$\frac{\partial \left(A(t)\right)}{\partial \beta(t)} = \min\left(0, c_i - c_j\right) e^{\beta(t)(c_i - c_j)}$$

(with $\frac{\partial w(t)}{\partial \beta(t')}$ being zero for all $t \neq t'$).

2.3 Descent With Variable Mutation Rate

As an alternative strategy to simulated annealing, we consider descent with a variable mutation rate. Each iteration of this heuristic consists of an attempt to "jump" to another state, which will be accepted if and only if the new state is of equal or lower cost. The size of the jump is a Poisson deviate, with expectation $\beta(t)$; thus the parameters $\beta(t)$ are the mutation rates.

This heuristic could alternatively be described as a $(1+1)$ Evolutionary Algorithm, i.e. an EA with a population of one, producing one mutated [1] offspring per generation, with the best of the two surviving to the next generation. The $(1+1)$ EA has been analysed extensively, with expectation (or upper bounds thereof) calculated[7] for the average FPT of this algorithm on a variety of problems. In common with many heuristics, most of the existing work toward modelling this algorithm has been focussed on 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 hard problems.

As with simulated annealing, suppose m is the transition matrix for a single mutation; then we may construct from this a matrix M representing a random (Poisson-distributed) number of mutations:

$$\begin{aligned} M(t) &= \sum_{n=0}^{\infty} m^n \mathbb{P}\left(n|\beta(t)\right) \\ &= \sum_{n=0}^{\infty} m^n e^{-\beta(t)} \beta(t)^n / n! \end{aligned}$$

[1] It should be noted that our Poisson-distributed number of mutations is subtly different to the (binomial) distribution given by a per-locus mutation probability, and that the loci at which our mutations are applied are chosen *with* replacement. If the expected number of mutations is $O(1)$, then the effects of both these differences diminish as the string length grows.

$$
\begin{aligned}
&= e^{-\beta(t)} \sum_{n=0}^{\infty} \left(m\beta(t)\right)^n / n! \\
&= e^{-\beta(t)} e^{m\beta(t)} \\
&= e^{\beta(t)(m-I)}
\end{aligned}
$$

(To efficiently compute the exponential $e^{\beta(t)(m-I)}$, we expand in terms of the eigenvalues and eigenvectors of m. If λ_i are the eigenvalues of m and v is a matrix composed of the right-hand eigenvectors of m such that the ith column corresponds to the eigenvector with eigenvalue λ_i, then $e^{\beta(t)(m-I)} = v^{-1}Dv$ where $D_{ij} = \operatorname{diag}\left(e^{\beta(t)(\lambda_i - 1)}\right)$.)

As with our treatment of simulated annealing, we define an acceptance matrix, to be multiplied element-wise by M, given simply by $A_{ji} = [c_j \leq c_i]$ (we use the convention that a bracketed predicate evaluates to 1 if the predicate is true, 0 otherwise). Again, a rejected step results in the search remaining at the same point; we account for this by increasing the leading diagonal so that each column sums to one:

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

The first derivative is

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

where $\frac{\partial M(t)}{\partial \beta(t)} = (I - m)\, M$.

2.4 Coarse-Grained Schedules

The optimisation of a schedule of length T requires a minimisation over a space equivalent to $\mathbb{R}^T$. For large T this requires considerable computation, and may be prone to numerical error. In order to reduce the computational cost of the optimisation, a "coarse-graining" may be applied to the schedule.

Specifically, an annealing schedule may be broken into blocks of length b, over which temperature is constant. In other words, $\beta(t) = \alpha\left(\lfloor \frac{t}{b} \rfloor\right)$, so that the schedule $\beta(t)$ is effectively replaced by a schedule $\alpha(t)$ of length T/b.

Coarse-graining is implemented by replacing the matrices $w(t)$ with $w(t)^b$. Calculating the gradient is straightforward, but tedious; it is, for sufficiently large b, still faster to compute than the gradient of a sequence of b different matrices.

3 Cost Functions

The idea of an optimal parameter set is meaningless without a definition of optimality. We shall examine two different "cost functions"; for each, optimality is achieved when the "cost" is *minimised*. In §8 we consider another cost function, the average first passage time.

We assume that the only relevant statistics are execution time and solution quality, and thus knowing the solution quality as a function of time is sufficient. Furthermore, we identify the number of iterations performed with execution time; this is reasonable, provided the number of (and time taken for) cost evaluations at each iteration is constant.

3.1 Where-You-Are (WYA)

The WYA cost depends only on which state the algorithm is in after a specific number of iterations. Obviously, this has little relevance to the solution of real problems (unless elitism is built into the heuristic itself); it does however have the benefit of being very efficient to compute. The average WYA cost is given by

$$C_{wya} = \mathbf{c}^\top \left(\prod_{t=1}^{T} w(t) \right) \mathbf{p}_0 = \mathbf{c}^\top \left(w(T) \cdot w(T-1) \dots w(2) \cdot w(1) \right) \mathbf{p}_0$$

where $\mathbf{c}$ is a vector such that c_i is the cost of state i. 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 terms.

The derivative of C_{wya} with respect to the schedule $\beta(t)$ is given by

$$\frac{\partial C_{wya}}{\partial \beta(t)} = \mathbf{c}^\top \left(\prod_{i=t+1}^{T} w(i) \right) \frac{\partial w(t)}{\partial \beta(t)} \left(\prod_{i=1}^{t} w(i) \right) \mathbf{p}_0$$

3.2 Best-So-Far (BSF)

The BSF cost is that of the best state reached on any iteration of the algorithm; the final state is not important, as all visited states are considered. In most practical scenarios, the goal of a search is usually to find a good solution rather than to have the algorithm terminate in the best state; this measure of optimality captures that idea.

The implementation is more complex than WYA; we use a method introduced by [3], where the transition matrices are modified to absorb particular cost values. The probability of ever reaching a state of cost a or lower is denoted by $\mathbb{P}(a)$, and given by

$$\mathbb{P}(a) = \mathbf{m}^\top \left(\prod_{t=1}^{T} d_a(t) \right) \mathbf{p_0}$$

where $\mathbf{m}$ is a "mask" vector with

$$m_i = \begin{cases} 1 & c_i \leq a \\ 0 & c_i > a \end{cases}$$

and each element of $d_a(t)$ is given by

$$d_{a,ij}(t) = \begin{cases} \delta_{ij} & c_j \leq a \\ w_{ij}(t) & c_j > a \end{cases}.$$

Let l be a list of the n distinct costs in $\mathbf{c}$, in increasing order. The BSF cost is then

$$C_{bsf} = \mathbb{P}(l_1)\, l_1 + \sum_{i=2}^{n} \left(\mathbb{P}(l_i) - \mathbb{P}(l_{i-1}) \right) l_i.$$

4 Test Problems

We shall use a small selection of binary string problems on which to evaluate the search heuristics. The first is a toy problem with a highly symmetric structure, providing ease of evaluation and a very large number of local optima. The others are "real problems" insofar as they are (very small instances of) problems known to be NP-hard.

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

We shall use a toy problem known as the "hurdle problem" [8] as a test problem.

The search space for the n-bit hurdle problem is the set of all binary strings of length n. The cost for any given string x depends only on the Hamming

distance between that string and some "goal string" g, and is given by

$$c(x) = \lceil H(x,g)/2 \rceil - (H(x,g) \mod 2)/2$$

(see fig. 1). The Hamming distance is simply the number of bit positions in which the two strings differ. For strings x, y of length L, this is given by $H(x,y) = \sum_{i=0}^{L} [x_i = y_i]$.

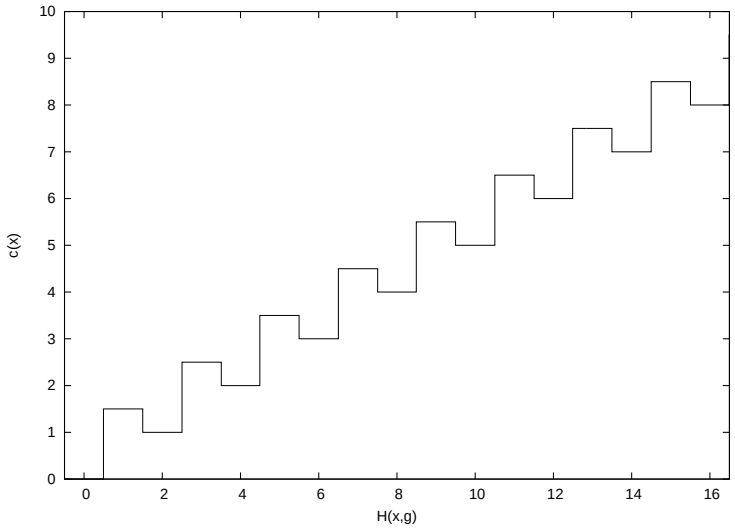

Fig. 1. Hurdle function cost vs hamming distance from goal string, for a 16-bit hurdle problem

If the string x is subjected to a single-bit mutation, then the new distribution of $H(x,g)$ can be computed entirely from the old distribution of $H(x,g)$. Therefore, the Hamming distance from the goal includes all the information we require, and we may reduce the search space of binary strings (2^n points) to a $(n+1)$-state model for $H(x,g)$.

This high degree of symmetry was one of the reasons for choosing to look at the hurdle problem; the other is that local minima are highly abundant in the search space, with exactly half of the points in the search space being minima (one of which is of course the global minimum). In fact, the search space for the n-bit hurdle problem is a bipartite graph, with the two vertex sets being the set of all the minima and the set of all the saddle points.

4.2 "Real" Problems

We shall also attempt to produce optimal schedules for a few small instances of "hard" problems. Three problem types were examined, each of which features a cost function defined over all binary strings of a particular length; in each case, this search space was reduced to a small number (usually 30-50) of states using a barrier tree model (§5).

Max-3-SAT The binary string is interpreted as a set of boolean variables. A set of "clauses" is created, each of which is the disjunction of three terms, each of which is either a variable or its negation. The cost of a string is the number of dissatisfied clauses.

Binary Perceptron A set of random data vectors is generated, where every entry is in $\{-1, 1\}$; each is also assigned a random label from $\{-1, 1\}$. A string is said to correctly classify a data vector if, when interpreted as a vector of this form, the scalar product with that data vector has the same sign as the label of that data vector. The cost of a string is taken to be the number of misclassified data vectors.

Spin Glass The cost can be thought of as a randomly-weighted sum of correlations between bits in the string. A random matrix J is generated, with each $J_{ij} \in \{-1, 1\}$. Again, a string x is interpreted as a set of values such that $x_i \in \{-1, 1\}$; the cost of the string is then given by $\sum_{i<j} J_{ij} x_i x_j$.

5 Reducing State Space for Large Problems

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 [9,10]. Note that we are constructing barrier tree 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.

5.1 Barrier Trees

Let S be a search space, with neighbourhood function $n : S \to 2^S$, and cost function $c : S \to \mathbb{R}$. An element y is said to be accessible from an element x if there exists a 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:

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

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. 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 $\exists x \in S_i, \exists y \in S_j, acc(x, y)$. (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 $S_j \equiv S_k$.

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 (example: fig. 2). 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. Each leaf node of the tree represents a minimum, while non-leaf nodes are saddle points.

(An alternative definition, that of level-connected sets, requires the path to have constant costs, and produces a considerably larger number of equivalence classes. The definition of connectedness between these classes then permits cycles, and thus a barrier *graph* is produced instead of a tree. To keep the size of the state space manageable, we shall look only at level-accessible sets.)

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 are 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. 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. 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. 3.

5.2 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 transition 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

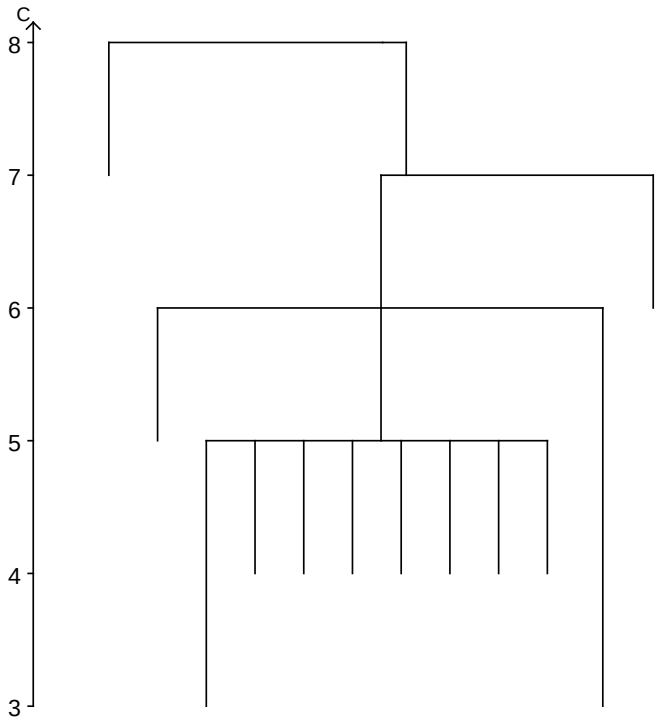

Fig. 2. An example of a barrier tree for a 20-variable Max-SAT problem. Only the area of the tree containing the global minimum is shown.

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, and we do not have a true Markov process; in assuming that this dependency can be ignored, we are making an approximation.

It is sometimes possible to construct a barrier tree for problems where the state space is too large to search exhaustively. A region around the global optimum of sufficient size to contain most of the local minima can be explored in full, with the rest of the points in the search space classified according to cost. The transition probabilities for these extra states may be estimated through random sampling of the search space[11].

6 Results

In order to find the optimal parameter set for any given search heuristic, we must perform a minimisation across the parameter space for that heuristic.

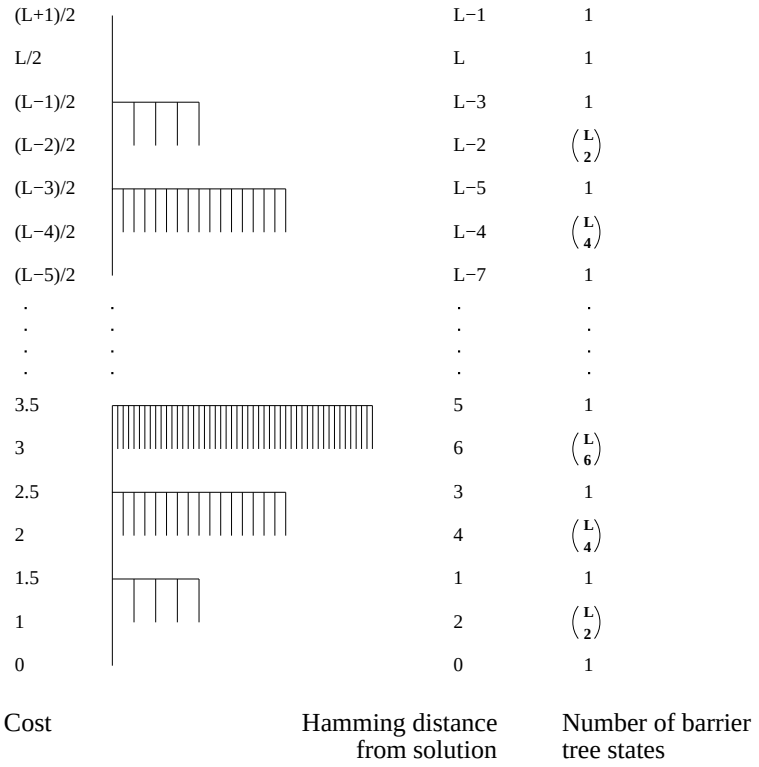

Fig. 3. Barrier tree states and costs for an L-bit hurdle problem. The tree pictured is illustrative rather than accurate for some L. Note that every point in the search space is either a local minimum or maximum. Because every local maximum is accessible from all other maxima of the same cost, the vast majority of the nodes correspond to local minima.

Simulated annealing and the descent with variable mutation 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. The length of the schedule is therefore the total number of iterations; we denote this by T.

We assume that the length of the schedule is fixed; the parameter space to be optimised over is then $\mathbb{R}^T$. Various gradient-based methods exist for continuous optimisation in high-dimensional space; we choose to use the Scaled Conjugate Gradient method (as implemented by [12]), with 500 iterations. This number of iterations was derived experimentally: those SCG runs which reached this limit had slowed to the point where further iterations produced no noticeable change.

The minimisations were repeated, beginning each time with a different random log-normally distributed schedule; no significant differences were observed in the optimised schedules. This is of course no guarantee that the parameter space is devoid of local minima, and we therefore cannot be completely certain that our schedules are the global optima.

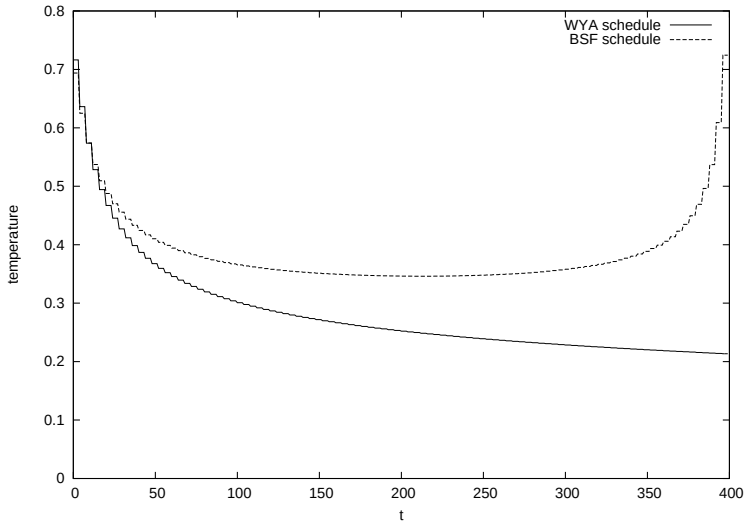

Fig. 4. Example WYA and BSF-optimised schedules for a 16-bit hurdle problem. Vertical axis is *temperature*, i.e. the reciprocal of $\beta(t)$.

6.1 Hurdle Problem

Schedules

Optimal annealing schedules were produced for the 16-bit hurdle problem. Fig. 4 shows one WYA-optimal and one BSF-optimal schedule (both of length 200); it is immediately apparent that they are qualitatively different.

Common to both schedules is an initial period of rapid cooling which then slows down; however, where the WYA schedule continues to cool for the remainder of the time, the BSF schedule actually increases temperature to a peak just before the end of the schedule. A probable explanation for this feature is that the BSF schedule is utilising the same strategy as WYA to arrive at a point which is likely to be close to the optimum, then performing a local search by attempting to visit as many neighbouring states as possibly (starting with those closest).

This behaviour prevents stagnation, as remaining in a low-cost state is of no particular benefit for a BSF algorithm, and doing so for too long will only impede exploration of the search space. By contrast, the WYA schedule promotes a gradual "settling", in order to finish as close to the optimum as possible.

A feature present in both schedules is a sudden decrease in temperature at the last step of the schedule. This is an obvious requirement for a good schedule; regardless of whether WYA or BSF cost is considered, the last step should never be in the uphill direction. The structure of the search space of the hurdle problem is such that every point is either a local minimum, or has only

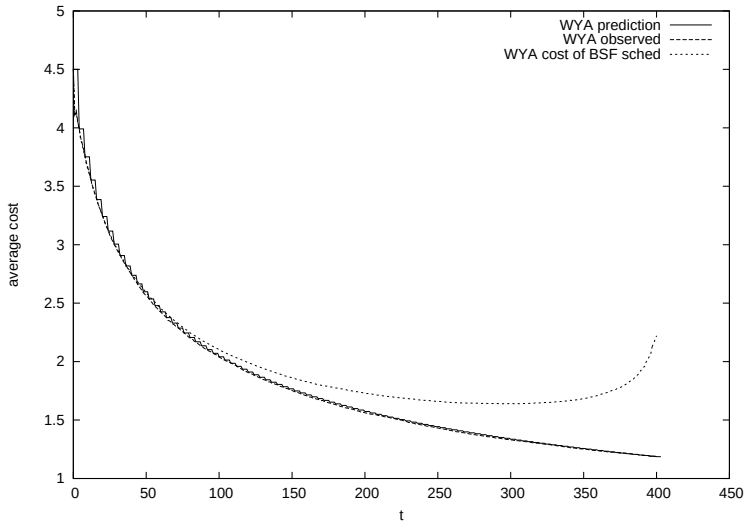

Fig. 5. Predicted vs. measured performance of a WYA schedule; measured results averaged over 100 000 runs.

local minima for neighbours; thus a single iteration at zero temperature will suffice to ensure that the algorithm terminates at a minimum. Although this is obviously of little direct importance for a BSF schedule, making an uphill move at the last iteration cannot possibly be of any use, whereas a downhill step could pay off in the event that the search is currently at a maximum, and one of the adjoining minima is of lower cost than any other visited thus far.

Performance

Figs. 5 and 6 show the performance of the WYA and BSF-optimised schedules respectively; the performance predicted by the model is almost indistinguishable from the observed performance in both cases. This agreement was to be expected, as the reduced model for the hurdle problem is an exact model of the original problem (§4.1).

The WYA evaluation of the BSF schedule (and vice versa) are shown for comparison: we observe that the WYA-optimised schedule also performs reasonably with BSF criteria (the converse is clearly not true).

6.2 Real problems

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. 7 was produced in this way, but has a very similar structure to those

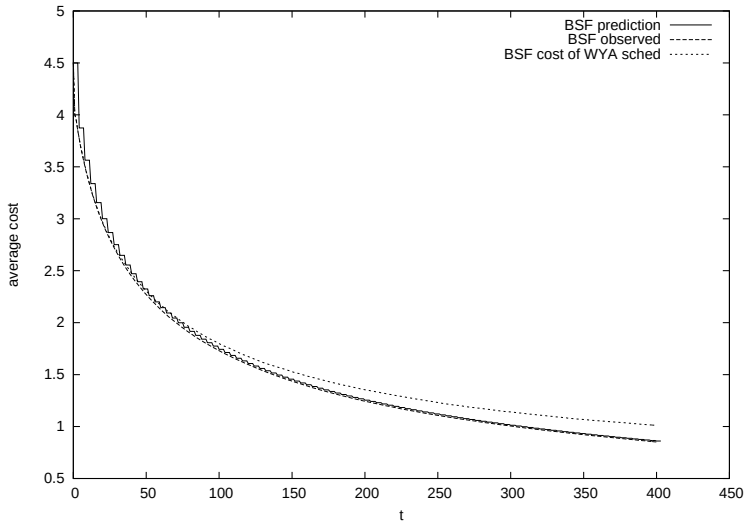

Fig. 6. Predicted vs. measured performance of a BSF schedule; measured results averaged over 100 000 runs.

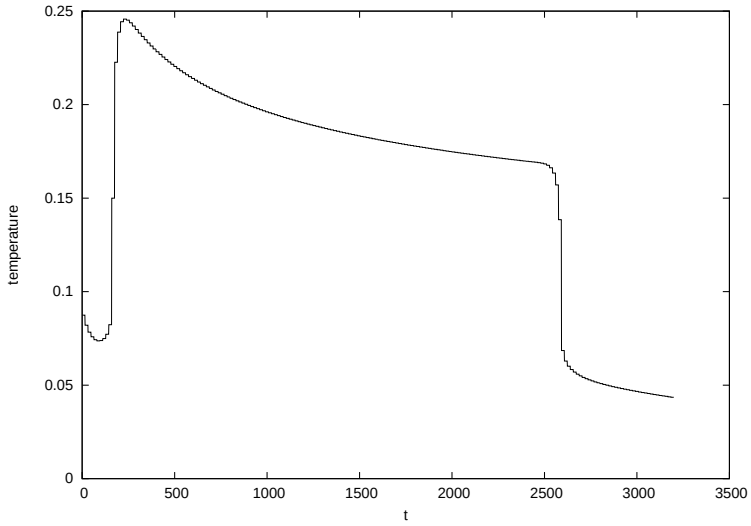

Fig. 7. An annealing schedule for a 40-variable Max-SAT problem, produced from a sampled barrier tree model with 85 states.

produced for the smaller problems.

On all of the "real" problems (Max-SAT, the binary perceptron, and the spin glass), it was found that, without exception, the predicted costs were underestimates; the optimised schedules did not perform so well in practise as the predictions of the model suggested. Fig. 8 shows the typical behaviour; the prediction is qualitatively correct, but over-optimistic.

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 (§5.2). However, the fact that the predicted performance is *always* over-estimated is worthy of investigation, for it suggests the presence

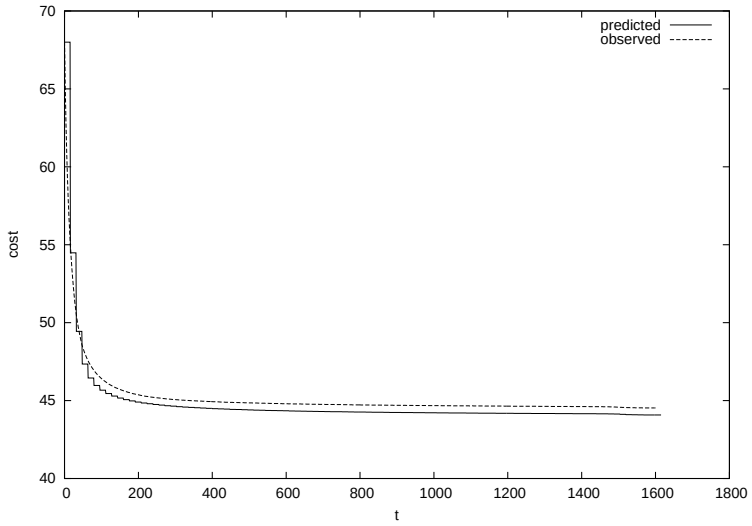

Fig. 8. Predicted vs observed performance of a WYA-optimal schedule for a spin-glass problem

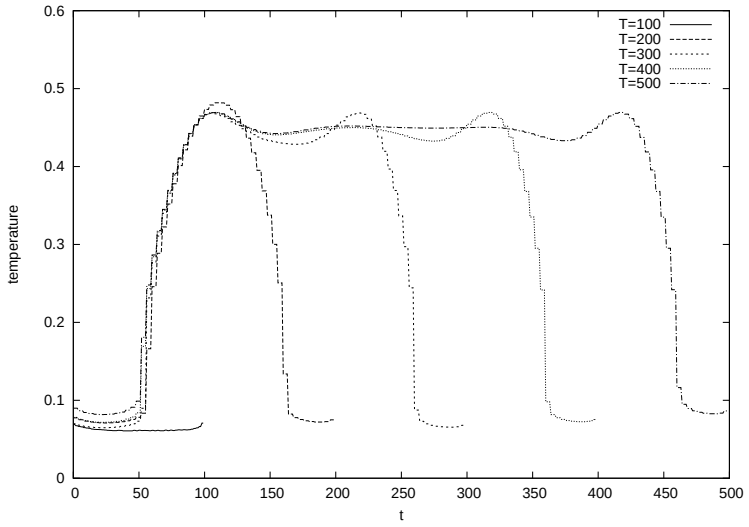

Fig. 9. BSF schedules on a Max-SAT problem were sometimes found to have periodic behaviour.

of a systematic effect by which the model problems become easier to solve than the problems themselves.

For some Max-SAT problems, BSF schedules displayed an oscillatory behaviour (fig. 9) which may be connected to that observed in [3]. This feature varied in prominence, with the deviation from the general shape (fig. 13) only being discernible in around half of the test problems. It is likely that the periodic behaviour effectively causes restarts in the search, producing 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.

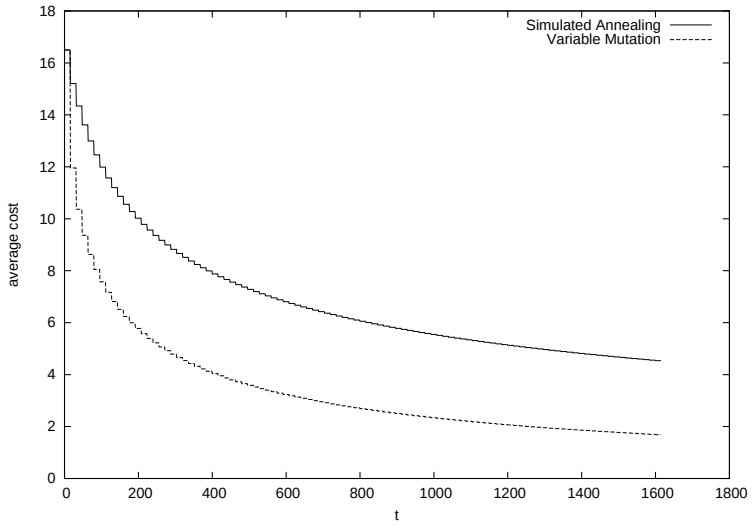

Fig. 10. Comparison between performance of optimised annealing and mutation schedules, on a 64-bit hurdle problem.

6.3 Variable Mutation

The variable mutation search was found to produce considerably better performance than simulated annealing on the hurdle problem (fig. 10), 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 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 [8].

Fig. 11 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 resembled the corresponding annealing schedules.

6.4 Coarse-graining and schedule length

In §2.4, we introduced a coarse-graining of annealing (or mutation) schedules with respect to time. Our motive was to produce a computational shortcut

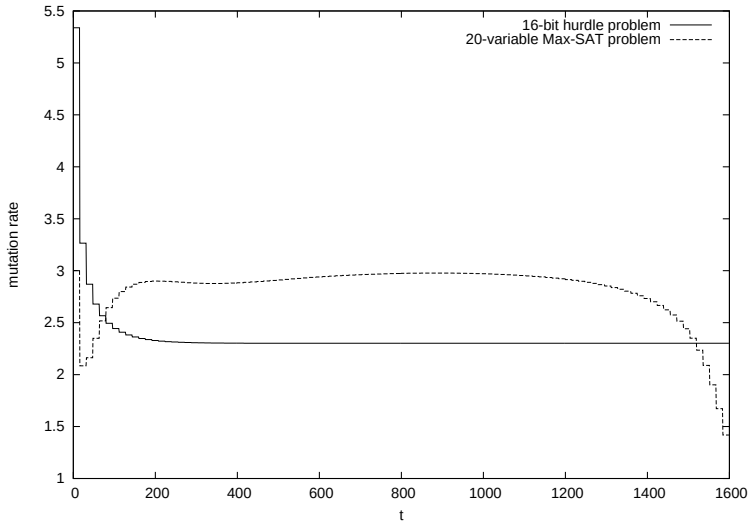

Fig. 11. Example mutation schedules for two problems.

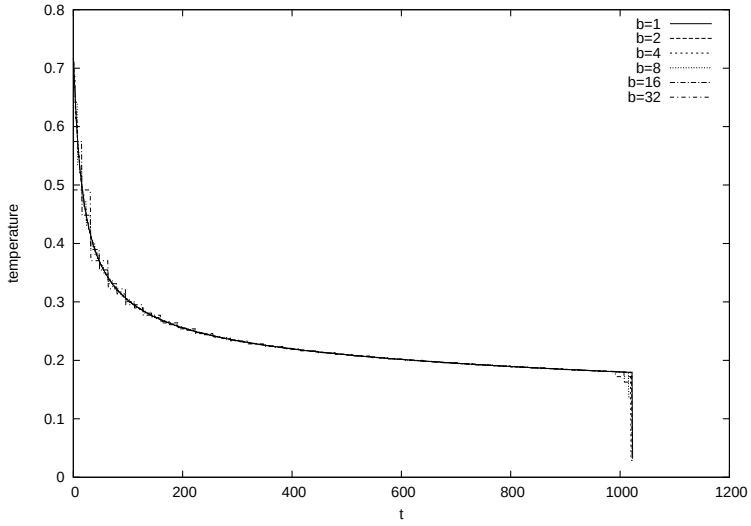

Fig. 12. Coarse-grained WYA schedules for a hurdle problem, various block sizes.

with which longer schedules could be studied. A variety of cases were examined; in each, a reasonable coarse-graining was found to produce a schedule which was representative of the original fine-grained one (e.g. fig. 12, 13).

Since very short schedules can be qualitatively different to longer ones, 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. However, it should be noted that annealing with coarse-grained schedules can be considered as simply a variant on simulated annealing. While providing a good approximation to "normal" schedules is certainly desirable, it is by no means essential.

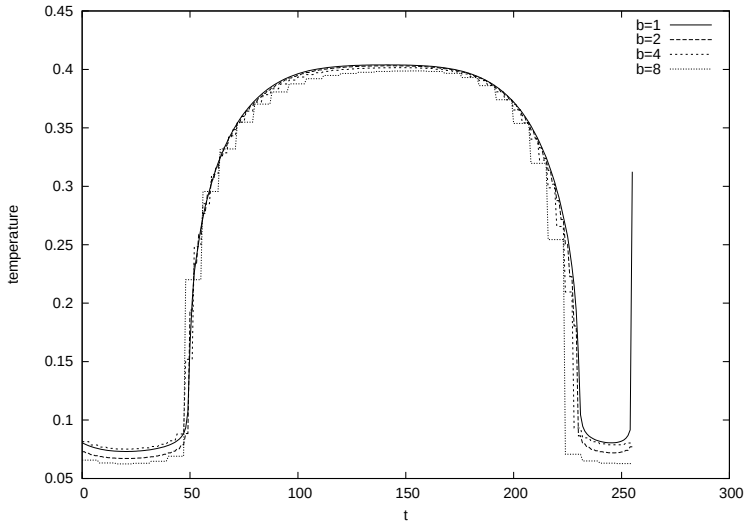

Fig. 13. Coarse-grained BSF schedules for a Max-SAT problem, various block sizes.

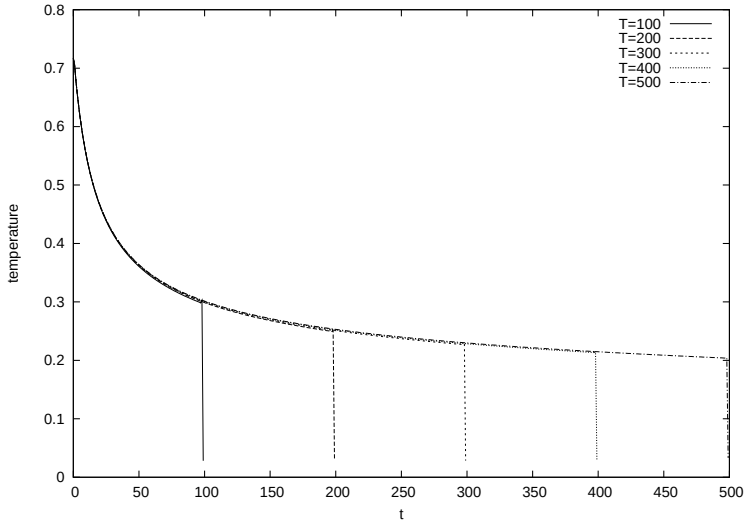

Fig. 14. WYA hurdle schedules show a very good overlap

6.5 Overlapping schedules

On some problems, schedules of different lengths were found to "overlap" significantly; ignoring behaviour at the extremities of the schedules, the curves were often so similar that they appeared coincident (fig. 14). This was found to be the case for the hurdle problem with WYA cost, and for some of the Max-SAT problems. Where this was not the case (e.g. fig. 15), the schedules showed much similarity, but were clearly distinct. It is possible that the degree of overlap may be connected to particular properties of the search space; in the case of the hurdle problem at least, it might be explained by the self-similarity of the cost landscape.

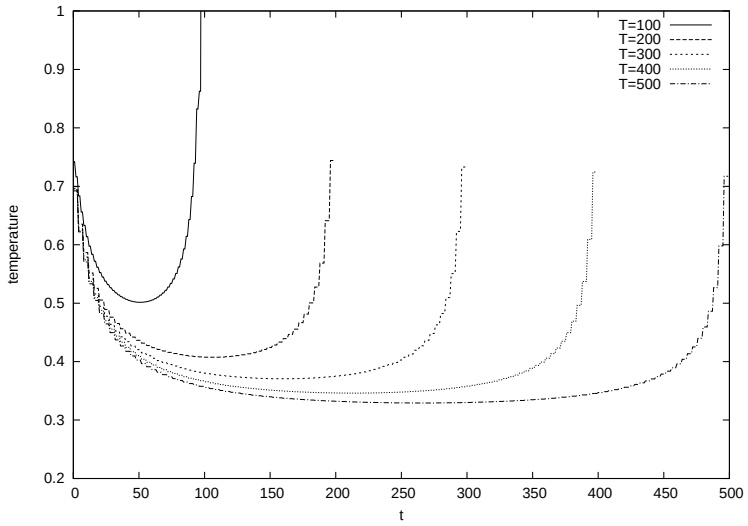

Fig. 15. BSF hurdle schedules show no overlap

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. 16 is an annealing schedule optimised 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 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 in-

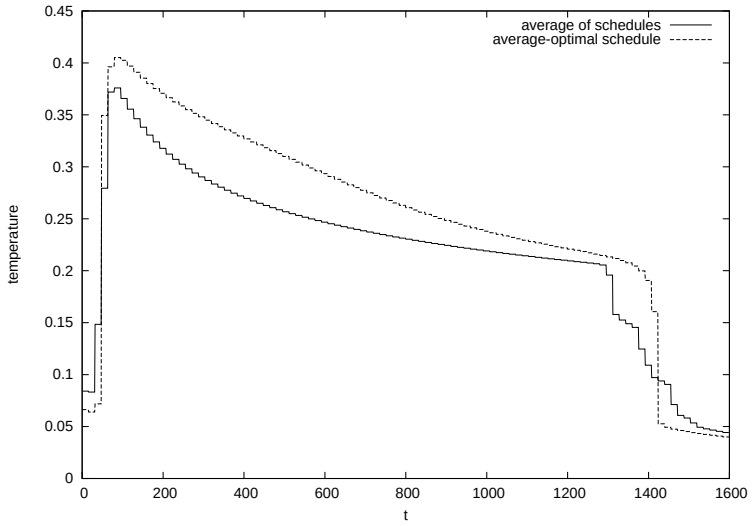

Fig. 16. Comparison between an average-optimal schedule and the average of all the individually-optimal schedules. Schedules produced for WYA cost of simulated annealing on a set of twelve Max-SAT problems; 100 (constant-temperature) blocks of 16 iterations.

dividual 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 (§6.2) and that the schedules were unlikely to be optimal, it was nonetheless unexpected that the average-optimal schedule should outperform the individual schedules (fig. 17) 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 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.

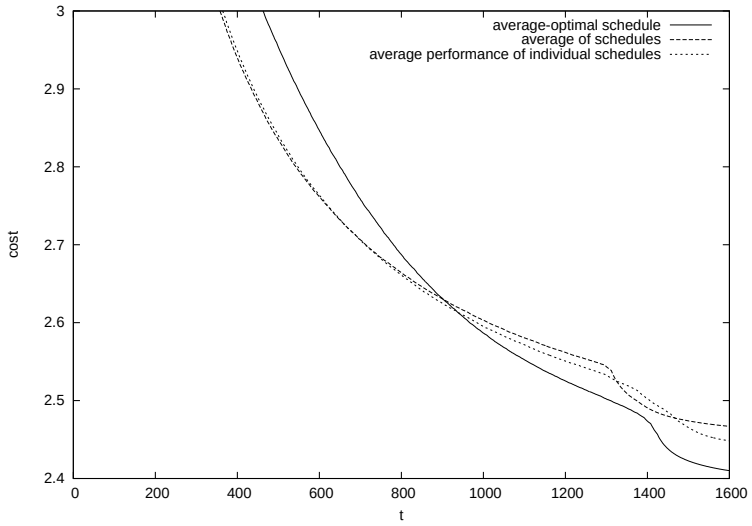

Fig. 17. When tested against the actual problems, the average-optimal schedule gives *better* performance than schedules specifically optimised for those problem instances.

8 Optimal Heuristics

In this section, we will examine several "toy heuristics" in an attempt to answer some of the questions raised in §6.2 regarding the inaccuracies in the predictions of our model. Our intention is to form a better understanding of the problems and the heuristics. In particular, we wish to know what information is required for the heuristics to be efficient.

For these heuristics, we shall use a different measure of cost: the average first passage time, defined to be the average number of iterations before the optimum is reached. Obviously, this is only well-defined for algorithms which are guaranteed to reach the optimum eventually (this is a necessary but not sufficient condition); those considered thus far have been time-dependent, and have been limited to a fixed number of iterations. However, the heuristics we are about to introduce are not time-dependent, and thus first passage time is relatively easy to compute [2]. There is also the additional advantage that no termination criterion is necessary; given a heuristic and a search problem, the best average FPT cost attainable is an absolute, unlike the WYA and BSF costs, for which the optimal parameters are themselves dependent upon the choice of termination criteria. Because of this, the FPT performance is commonly used in complexity analysis.

[2] By this we mean that the average FPT is given by a simple, closed-form expression, requiring solution of one linear equation. The calculation requires $O\left(n^3\right)$ operations (where n is the number of states), and thus grows faster with n than WYA or BSF cost ($O\left(n^2 T\right)$ and $O\left(|\mathcal{C}|\, n^2 T\right)$ respectively, where T is the number of heuristic iterations and $\mathcal{C}$ is the set of costs).

8.1 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. This may be calculated with the assistance of a modified transition matrix $\hat{w}_{ij} = (1 - e_j)w_{ij}$, where $e_i = [\forall j, c_i \leq c_j]$, so that $\hat{w}$ is identical to w with the exception that the columns corresponding to the transition probabilities *from* the optimal state(s) are replaced with columns of zeros. Note that $\hat{w}$ is no longer a stochastic matrix, and is in fact singular.

The probability of a particular first passage time t occurring is $\mathbb{P}(T_{fp} = t) = \mathbf{e}^\top \hat{w}^t \mathbf{p}_0$; we note that, over all t, these probabilities must sum to one, and thus

$$\mathbf{e}^\top \left(\sum_{t=0}^{\infty} \hat{w}^t \right) \mathbf{p}_0 \quad = \quad \mathbf{e}^\top (I - \hat{w})^{-1} \mathbf{p}_0 \quad = \quad 1$$

Since this must be true *for all* $\mathbf{p}_0$, we have $\mathbf{e}^\top (I - \hat{w})^{-1} = \mathbf{1}^\top$, and therefore we can replace the symbol '$\mathbf{e}^\top$' with $\mathbf{1}^\top (I - \hat{w})$.

$$\begin{aligned}
\langle T_{fp} \rangle &= \sum_{t=0}^{\infty} t \mathbb{P}(T_{fp} = t) \\
&= \sum_{t=0}^{\infty} (t+1) \mathbb{P}(T_{fp} = t) - \sum_{t=0}^{\infty} \mathbb{P}(T_{fp} = t) \\
&= \mathbf{e}^\top \left(\sum_{t=0}^{\infty} (t+1) \hat{w}^t \right) \mathbf{p}_0 - 1 \\
&= \mathbf{e}^\top \frac{\mathrm{d}}{\mathrm{d}\hat{w}} \sum_{t=1}^{\infty} \hat{w}^t \mathbf{p}_0 - 1 \\
&= \mathbf{1}^\top (I - \hat{w}) \frac{\mathrm{d}}{\mathrm{d}\hat{w}} (I - \hat{w})^{-1} \mathbf{p}_0 - 1 \\
&= \mathbf{1}^\top (I - \hat{w})^{-1} \mathbf{p}_0 - 1
\end{aligned}$$

8.2 Lower Bound Algorithm

It is possible to devise a heuristic with full a-priori knowledge of the search space. The "lower bound algorithm" is the optimal such heuristic one can produce without violating the following constraints:

- 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 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 reducible to 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.

Thus our $\mathbb{P}_A(i \to j)$ will 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 §2.2 and §2.3. The optimal acceptance matrix is produced 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.

8.3 Cost-Dependent Algorithm

We now impose one further restriction on the search heuristic: the only information available about a state shall be the cost; thus states of equal cost are indistinguishable. A randomly chosen neighbour which happens to have lower cost may be a local minimum, or may lead to the global minimum - there is no way for the algorithm to tell, and thus the algorithm need no longer be time-independent and deterministic. The latter we accept, permitting the acceptance probabilities to take fractional values; the former we constrain artificially for reasons of practicality. Our goal is to determine, by comparison to the lower bound algorithm, whether the loss of information about individual states is to the detriment of the heuristic's performance.

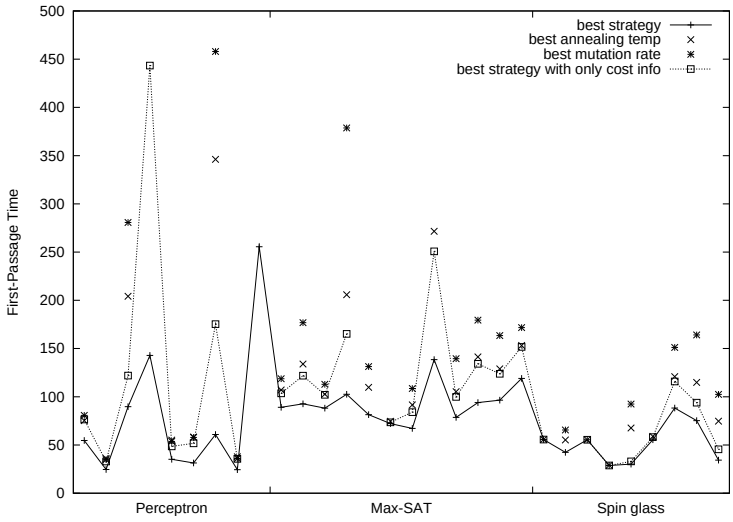

Fig. 18. FPT for four different algorithms on (barrier trees of) a variety of test problems. Within each category, problems are sorted by number of barrier tree states. On a small number of problems, the optimisation for the cost-dependent algorithm consistently diverged, hence the occasional missing points from this series. The other absent data points are those few results with times exceeding 500 iterations.

8.4 Single-Parameter Heuristics

FPT is difficult to calculate for the heuristics in §2, due to the implicit assumption of a finite number of iterations. We avoid this issue by introducing single-parameter annealing, where the annealing temperature is held constant across all time; an identical simplification is used for the mutation search. (It is indeed possible to use an alternative parameterisation of the annealing/mutation schedules which allows for infinite schedules to be represented by a fixed number of parameters.)

8.5 FPT Results

Average first passage times were computed for each algorithm on a set of barrier tree problems, and are shown in fig. 18. It was frequently found that the matrix $(I - \hat{w})$ was near-singular; computation of derivatives was especially awkward, so a method not requiring an explicit gradient (the Nelder-Mead Simplex method) was favoured instead. The optimisations were repeated ten times for each problem, 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 algo-

rithm, 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 §8.2. 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 scale of the problems we examined.

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 (§6.3, [8]). 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.

9 Conclusions

In this paper we have constructed a general framework for optimising parameters (specifically, annealing and mutation schedules) for searches on problems too large to be analysed with existing methods. Although the problems we examined were of trivial size compared to those typically used as benchmarks (which are admittedly still too large for us to analyse), they were considerably

larger than the toy problems previously studied. Search spaces with millions of elements were reduced to a barrier tree model with around 30-50 states, transforming the parameter optimisation process from an infeasibly vast computation to a task easily achievable with quite modest resources.

The strength of the approach is that it may be applied to any finite optimisation problem, since a barrier tree can be constructed for any finite cost landscape. The caveat is that the barrier tree fails to capture all the relevant information from the original search space. This is of course to be expected for any "hard" problem.

Consequently, the schedules optimised on the model are not optimal on the problems themselves; real performance is consistently poorer than that predicted by the model, and it is not difficult to find better schedules on the real problem. (However, the qualitative agreement between the model and the real behaviour is good.)

Although the inaccuracy of the predictions means that this approach is unlikely to be of direct use to practitioners, it may be of substantial theoretical benefit when attempting to understand why search algorithms struggle with "hard" problems. Where we might previously have assumed that the complexity lay entirely within the structure of local minima and barriers, we now know that this is not true. Furthermore, we have at least a partial model of a search on a hard problem; there are possibilities for studying the causes of the discrepancies and producing refinements to the model.

There is potential for average-optimal schedules to be applied to scenarios where large groups of similar problems are presented, although the resulting increase in effectiveness must obviously be weighed against the computational cost of producing such a schedule.

In §8 we calculated average first passage time for simplified single-parameter versions of simulated annealing and variable mutation search; the former was found to consistently outperform the latter, in contrast with the observations for WYA cost for the original versions of the heuristics (§6.3). Within the constraints of §8.2, an optimal algorithm was formulated, which provided a lower bound for the FPT of single mutation time-independent heuristics such as constant temperature simulated annealing. The mutation search did not violate this bound, despite not meeting the single mutation criterion.

References

[1] K. H. Hoffmann, P. Salamon, The optimal simulated annealing schedule for a simple model, J. Phys. A: Math. Gen. 23 (1990) 3511–3523.

[2] M. Christoph, K. H. Hoffmann, Scaling behaviour of optimal simulated annealing schedules, J. Phys. A: Math. Gen. 26 (1993) 3267–3277.

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

[4] P. N. Strenski, S. Kirkpatrick, Analysis of finite length annealing schedules, Algorithmica 6 (1991) 346–366.

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

[6] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

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

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

[9] J. Hallam, A. Prügel-Bennett, Crossover and barrier based markov models, in: Proceedings of CEC 2005, Vol. 2, 2005, pp. 1661–1666.

[10] J. Hallam, A. Prügel-Bennett, Constructing model problems based on hard optimisation problems, submitted to Theoretical Computer Science (2005).

[11] J. Hallam, A. Prügel-Bennett, Large barrier trees for studying search, IEEE Transactions on Evolutionary Computation 9 (4) (2005) 385–397.

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