UNIVERSITY OF SOUTHAMPTON

Barrier Trees For Studying Search Landscapes

by

Jonathan Hallam

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

July 2006

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

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

Doctor of Philosophy

by Jonathan Hallam

The landscapes of Combinatorial optimisation problems are huge, multidimensional, topographically complex structures. A better understanding of the cost landscapes of difficult Combinatorial optimisation problems is vitally important to the development of a strong theory to guide the design of heuristic search algorithms. This thesis presents Barrier Trees and Barrier based models as tools to study small instances of problems exempla of the difficult problems faced by practitioners.

Barrier Trees represent a cost landscape as a tree structure, with the leaves representing local minima and the internal nodes representing barriers between different parts of the landscape. They are a development of similar structures used in the study of protein folding, molecular clusters and potential energy surfaces amongst other fields. A novel definition of Barrier Trees is developed which has the advantage of defining a partitioning of the landscape, which can be used for further analysis and the animation of search heuristics working on the landscape. These techniques and various analyses using this partitioning are demonstrated.

Barrier based models are model problems derived from small instances of real problems. They maintain the structure and connectivity between different local minima, and many other properties of the original problem, while massively reducing the state space. They provide a model Neighbourhood function that allows many different search algorithms to be implemented on the model problem, and they are particularly well suited to modelling search heuristics as Markov chains. These models are defined, and simple descent is used as an example of how the models can be used. The relationship between descent on the model problem and on the original problems is examined in detail. Variations on the basic model are explored.

Contents

List of Figures

List of Tables

Listings

Nomenclature

$\mathcal{P}$,$\mathcal{S}$,$\mathcal{C}$	Sets are given a capital, calligraphic character.
P_1,S_2,C_2	When indexing a singular members from a set, the set is written in a normal font.
$\mathbf{M}$,$\mathbf{T}$,$\mathbf{A}$	Matrices have a capital, bold character.
$M_{1,1}$,$T_{2,1}$,$A_{x,y}$	When indexing a singular member of a matrix the matrix letter is written in a normal font
$\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$	Vectors have lower case bold letter. All vectors are column vectors
x_1,y_2,z_3	When indexing a singular member from a vector, the vector is given a normal font.
$\|\mathcal{S}\|$	The cardinality of set $\mathcal{S}$.
$2^{\mathcal{S}}$	The power set of set $\mathcal{S}$.
δxy	Kronecker delta function. 1 if $x = y$, 0 otherwise.
$[expr]_b^a$	An indicator function that evaluates to a if $expr$ is true and to b if $expr$ is false.
$\boldsymbol{\delta}_n$	The vector consisting of 0s, except at index n which is 1.
$\mathbf{I}$	The identity matrix.
$\mathbf{1}$	The vector of all 1s.
$f(\boldsymbol{x})$	The cost function of a problem, $f : \mathcal{C} \mapsto \mathbb{R}$
$N(\boldsymbol{x})$	The Neighbourhood function of a problem, $N : \mathcal{C} \mapsto 2^{\mathcal{C}}$
π	The set of all paths on a landscape.

Chapter 1

Introduction

This thesis presents novel tools for studying heuristic search algorithms on Combinatorial optimisation problems. Combinatorial optimisation problems are extremely common in any number of practical situations. Many resources are by their nature discrete, such as machine parts, desk spaces and aircraft crew, and organising these discrete components efficiently often requires the solution of a Combinatorial optimisation problem. In real life situations, these problems often involve a huge number of resources, and even a small improvement in the quality of the solution found can save many millions of pounds.

As well as being extremely common, Combinatorial optimisation problems are often computationally difficult. In fact, Combinatorial optimisation problems are often examples of problems that are '$\mathcal{NP}$-hard', the classic class of computationally difficult problems. It is believed that these problems take exponentially longer to solve exactly as the size of the problems grows, so that increasing computer power is of limited use. Not all Combinatorial problems can be definitely classed in this category, but the size and complex set ups of many problems make exact algorithms impractical.

Heuristic search algorithms are one of the most popular approaches to solving Combinatorial optimisation problems. They can be easily applied to a wide range of problems, and require very little *a priori* knowledge of the problem class. For many problems with complex constraints and goals they are the only practical type of algorithm to use, as interpreting the problem in greater detail is impractical. The stochastic nature of search algorithms means it is often worthwhile repeating them several times to increase the chance of a good result; this makes it extremely easy to exploit parallel computing resources.

Despite their wide use and a great deal of attention from the theoretical community, heuristic search algorithms have proven to be extremely challenging to analyse

theoretically. There is no well established methodology into how heuristic search algorithms should be designed, little understanding of their behaviour or what makes problems hard for heuristic search algorithms. Considering the importance of Combinatorial optimisation, and the wide use of Heuristic search algorithms to solve them, such a theory would be extremely useful. As Papadimitriou states:

> developing the mathematical methodology for explaining and predicting the performance of heuristics is one of the most important challenges facing the fields of optimisation and algorithms today (Papadimitriou and Steiglitz, 1998).

A major challenge to the study of heuristic search algorithms is the difficulty with working with complex Combinatorial optimisation problems. In the context of heuristic search it is common to describe problem instances in terms of a 'cost landscape' metaphor. The cost landscapes of problems of interest to practitioners are typically huge, highly dimensional with complex topologies. Working with these structures is so difficult that many basic questions about them are still unknown. Faced with this difficulty many theoreticians have developed model problems that have some of the features of challenging optimisation problems but are far more amenable to mathematical analysis; typically the model problems have a large degree of symmetry that can be exploited. The hope is that these model problems can teach us something about heuristic search which can then be applied to the more challenging problems faced by practitioners. So far this approach has not succeeded. One of the central themes of the recently developed 'No Free Lunch' theorems is the importance of knowing what features of a problem class a search heuristic tries to exploit. Unfortunately, the challenges of studying instances of Combinatorial optimisation problems means that their exploitable structure is poorly understood.

The work in this thesis develops Barrier Trees as a tool to study and characterise problem instances. Barrier Trees represent a problem instance as a tree structure, which can be used as both a visualisation of the landscape and as the basis of further analysis. Cost landscapes of practical problems are intractably difficult to analyse; Barrier Trees only overcome this difficulty because they are limited to small problem instances. However, these small instances still have a direct relationship to more challenging practical problems, and the technique of extrapolating from small systems to larger ones, scaling analysis, is well understood.

Barrier Trees are well established in the field of physics and biology, and have been used to study Combinatorial optimisation problems. The contribution of this Thesis is a new (though equivalent) definition of Barrier trees, which has the benefit of providing a mapping of every configuration to some part of the tree. This mapping

greatly increases the usefulness of Barrier Trees, providing information that can be used for many analyses that are not possible without it. This mapping can also be used to animate a heuristic search algorithm acting on the problem; this technique along with several other uses of the mapping information are presented. A more complex use of this information is the development of model problems which we call 'Barrier-based' models. This is a powerful technique for creating model problems from real problem instances that are amenable to mathematical analysis. A definition of these models and an analysis of their accuracy makes up the second part of this thesis.

All the techniques presented in this Thesis have been implemented by me. The ideas have mostly been developed in discussions with my tutor Adam Prügel-Bennett. The work in chapters 4 and 5 has been previously presented in Hallam and Prügel-Bennett (2003, 2005c). The Barrier-based models introduced in chapter 6, the results presented in chapter 7 and the improvements investigated in chapter 8 with the exception of the work on crossover have been submitted to *Theoretical Computer Science* as Hallam and Prügel-Bennett (2005b). The work on crossover has been presented in Hallam and Prügel-Bennett (2005a).

Chapter 2

Combinatorial Optimisation

This chapter provides a brief background on Combinatorial optimisation, the problems that are the focus of this thesis. This starts with a brief formalisation of combinatorial optimisation problems. This is followed by a description of the theory of $\mathcal{NP}$-completeness, one of the most important concepts in complexity theory, and a brief discussion of some of its implications. The concept of phase transitions and their relationship to the complexity of $\mathcal{NP}$-complete problems is then discussed. This chapter concludes with some brief descriptions of several exemplar problem classes.

2.1 Notation

A brief note on the mathematical notation that will be used.

- Atomic objects and functions will have a normal lower-case letter e.g. $x, y, f(x)$.
- Vectors and strings will be have a lower-case bold symbol e.g. $\boldsymbol{x}$,$\boldsymbol{y}$ and $\boldsymbol{z}$. Vectors will always be column vectors.
- Sets will in general be represented by a bold, calligraphic font, e.g. $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. Due to the variety of mathematical constructs that can be considered sets, this is somewhat arbitrary.
- Matrices are indicated by a bold capital letter, e.g. $\mathbf{M}$.
- When an atomic member of a vector, set or matrix is indexed, the vector, set or matrix is written in a normal font. This can be useful for clarifying the meaning of certain complex expressions e.g. $\mathbf{M}^t_{x,y}$, the x, y element of the matrix $\mathbf{M}^t$, or $(\mathbf{M}^t)_{x,y}$, as opposed to $M^t_{x,y}$, the x, y element raised to the power t of $\mathbf{M}$, or $(\mathbf{M}_{x,y})^t$.

2.2 Combinatorics

In almost every industry and service, resources which are indivisible, whether they are labourers, machinery parts or work shifts need to be organised to meet some criteria. These organisation problems are practical examples of Combinatorial Optimisation problems. The huge range and importance of Combinatorial Optimisation cannot be over-stated. These problems are often considered as the classic example of "computationally difficult" problems, and any improvements in the techniques used to solve them could lead to large rewards.

The general form of a Combinatorial Optimisation problem is simple. An optimisation problem requires that the arguments that minimise or maximise the value of some function are found. In a *Combinatorial* Optimisation problem, the domain of the function is a set of combinations of the input variables.

Formally, an instance of a Combinatorial Optimisation problem can be defined as having,

1. A discrete set, $\mathcal{C}$, of feasible solutions,
2. An objective function, sometimes called a fitness or cost function, $f : \mathcal{C} \rightarrow \mathbb{R}$,
3. An optimum, normally the minimum or maximum.

Often, a feasibility predicate is used to define the set of feasible solutions as a subset of some larger solution set. This is particularity relevant when finding feasible solutions is expensive. For our purposes, we consider infeasible solutions as having an infinite cost (or negative infinity fitness) within the above framework. Note that the difference between minimisation or maximisation is trivial, as the maximum of $f(\boldsymbol{x})$ is the minimum of $-f(\boldsymbol{x})$. Throughout this thesis we will consider minimisation, so that our objective function is a cost function and a lower cost solution is the better solution.

From the perspective of algorithm design, algorithms are created to solve *instances* of a particular *problem class*. A problem class specifies a general form for many different instances, each of which has a different cost function and different optimal solutions. An instance is specified by some additional information. For example, an algorithm could be designed to arrange different shaped boxes to fit inside the smallest possible containing box. This would be considered a class of problems (this is in fact a version of the 'knapsack' problem class). A particular set of boxes of different sizes is the additional information required to specify an *instance* of the problem with some specific set of optimal solutions. An algorithm designed to solve the class of box packing problems could then be applied to many different instances. Some examples of some well known problem classes are given in section 2.3.3.

2.3 Problem Difficulty

In studies of complexity, a difficult problem is one which requires a large amount of some limited computational resource. It is not related to the human intellectual effort required to solve, or create an algorithm to solve, a particular problem. Usually, the resource considered is runtime, the amount of time required on a computer to solve the problem, although sometimes other resources such as computer memory are considered. In practice, one resource can often be exchanged for another, for example, caching techniques can be used to reduce the runtime at the cost of using more memory.

In general, there is no method of proving that a particular problem class *requires* any amount of a particular resource to solve. A problem's difficulty is given as the resources required by the best known algorithm to solve it. In some rare cases it is possible to obtain lower bounds on the resources needed to solve a problem, but for most problems this is not known. For most problem classes, it is possible that there is a currently unknown algorithm that could solve the problem class using less resources. Computational complexity theory uses several techniques to work around this limitation, and some of the most important results are described in 2.3.1.

Due to the rapidly increasing nature of computer performance, and the vast differences that can occur between different implementations of the same algorithm, an algorithm's performance is described by the way it grows as the problem size increases. Only the dominant term of the function describing the algorithm's performance as the size of a problem increases to infinity is considered, the asymptotic value. This is written using "Big-O" notation, so that an algorithm that takes $4n^2 + 10$ seconds to solve a problem, (where n is a measure of problem size), is said to take time $O(n^2)$. Two important groups of algorithms are those said to have polynomial or better run times, and those said to have worse than polynomial run times. Polynomial algorithms are those with run times such as $O(n^2)$ or $O(\log n)$; examples of worse than polynomial algorithms are those with exponential run times, $O(e^n)$, or factorial runtimes $O(n!)$.

2.3.1 $\mathcal{NP}$-Hardness

It is, unfortunately, extremely difficult to show what the best possible algorithm for a particular problem class is, and this limits our ability to specify 'hard' and 'easy' problem classes. Complexity theory has developed several different techniques to partially overcome this limitation, and several *complexity classes* of different problems have been defined. Of these, the class of $\mathcal{NP}$-hard problems is the most important.

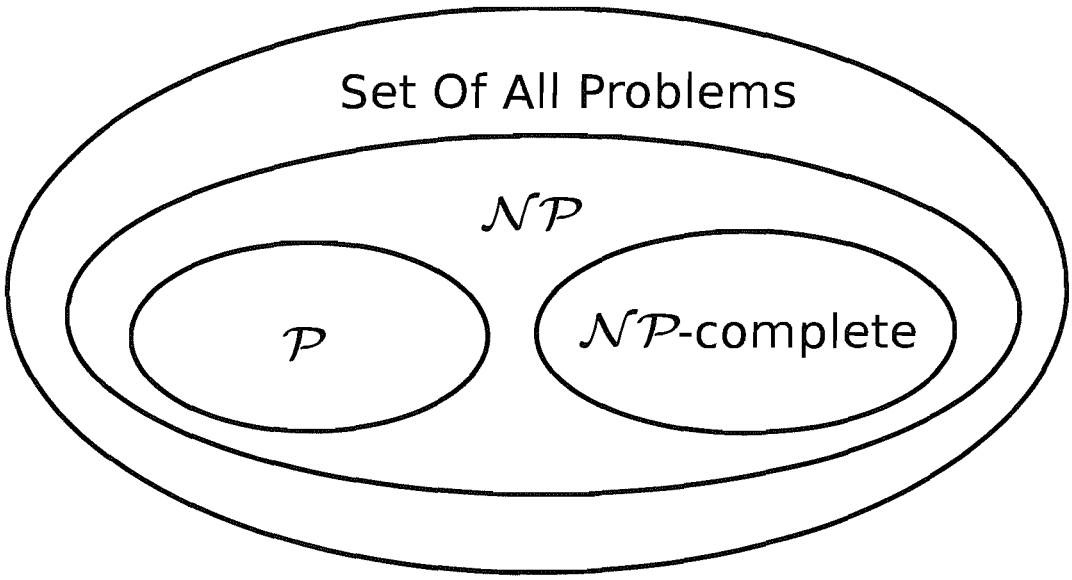


FIGURE 2.1: Venn diagram showing the problem complexity classes of $\mathcal{P}$, $\mathcal{NP}$ and $\mathcal{NP}$-complete, assuming $\mathcal{P} \neq \mathcal{NP}$

To define $\mathcal{NP}$-hard problems, it is first necessary to discuss the problem classes $\mathcal{P}$ and $\mathcal{NP}$. $\mathcal{P}$ and $\mathcal{NP}$ are complexity classes of *decision* problems. Decision problems have a simple 'yes' or 'no' answer. The complexity of decision problems is simpler to analyse than other types of problem. In particular, it is simpler to *reduce* problems to decision problems, which will be described shortly. The difference between the complexity classes $\mathcal{P}$ and $\mathcal{NP}$ is on the types of Turing machine in which they can be solved in polynomial time. Turing machines are theoretical computing machines used to analyse automatic computation. A deterministic Turing machine is closely related to modern computers. A non-deterministic Turing machine is purely theoretical; in essence, a non-deterministic Turing machine never has to make decisions without the necessary information to make the decision, while a deterministic Turing machine has to try every possible solution. The complexity class $\mathcal{P}$ consists of decision problems with a known deterministic-polynomial time algorithm. The problems in $\mathcal{NP}$ have a non-deterministic polynomial time algorithm. Note that deterministic algorithms take the same or less time on a non-deterministic machine, so $\mathcal{P} \subset \mathcal{NP}$ (see figure 2.1).

Reduction is a central concept in defining complexity classes. Reduction makes it possible to state that one problem class is *at least as* difficult as another. As already mentioned, reduction is simpler to define on decision problems. If we have two decision problem classes $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ is reducible to $\mathcal{B}$ if

1. There exists a transform, from every instance in $\mathcal{A}$ to an instance in $\mathcal{B}$. We label this transform as a function $T : \mathcal{A} \mapsto \mathcal{B}$.
2. T can be performed in deterministic polynomial time.
3. $T(A_i) \in \mathcal{B}$ has the solution 'yes' if and only if A_i has a solution of 'yes'.

This is the specific form of reduction used to define the class of $\mathcal{NP}$-complete problems. In complexity theory, reduction has a more general definition that encompasses problems other than decision problems, different resources than runtime and different boundaries than polynomial and worse than polynomial. For our purposes, reduction as it has been defined here is sufficient.

The way reduction is used to define difficulty often causes confusion, because it involves the use of the contra-positive rule. To describe how it is used, we define reduction as a relationship $R(\mathcal{A}, \mathcal{B})$ which is true if and only if $\mathcal{A}$ reduces to $\mathcal{B}$, and we define $E(\mathcal{A})$ as a predicate that is true if there exists a polynomial time algorithm to solve (A). $E(\mathcal{A})$ therefore matches our concept of an easy problem, and $\neg E(\mathcal{A})$ indicates a difficult problem. We can then state the axiom

$$R(\mathcal{A}, \mathcal{B}) \wedge E(\mathcal{B}) \rightarrow E(\mathcal{A}).$$

In words, if $\mathcal{A}$ reduces to $\mathcal{B}$, and $\mathcal{B}$ is 'easy', then $\mathcal{A}$ is easy. This follows because if $\mathcal{A}$ reduces to $\mathcal{B}$ and there is a polynomial time algorithm to solve $\mathcal{B}$, then any instance of $\mathcal{A}$ can be solved by transforming it into an instance of $\mathcal{B}$ and solving it using the algorithm for $\mathcal{B}$, all in polynomial time. We are more interested in showing difficulty, so we express the second part using the contra-positive rule,

$$R(\mathcal{A}, \mathcal{B}) \wedge \neg E(\mathcal{A}) \rightarrow \neg E(\mathcal{B}).$$

This is the statement that the more usual expression of reduction comes from: if $\mathcal{A}$ reduces to $\mathcal{B}$ then $\mathcal{B}$ is as difficult as $\mathcal{A}$. This is an extremely specific use of the word difficult.

The central discovery in defining $\mathcal{NP}$-hard problems was made by Stephen Cook in Cook (1971). He showed that the problem of running a non-deterministic Turing machine could be reduced to solving a SAT problem. Therefore, if SAT could be solved in deterministic polynomial time, any problem in $\mathcal{NP}$ could also be solved deterministically, so $\mathcal{P} = \mathcal{NP}$. This result was followed up by Karp (1972) showing that 21 well known difficult problems were of equivalent difficulty to SAT (i.e. the problem reduced to SAT and SAT could be reduced to the problem). This group of problems is called $\mathcal{NP}$-complete, and has been continuously added to until now

there are several thousand problems that have been shown to be members (Garey and Johnson, 1979).

Decision problems have little direct relevance to most practical applications. To show that other types of problems are difficult, a $\mathcal{NP}$-complete decision problem that can be reduced to the problem in question is found. If such a decision problem can be found, the problem is as difficult as the $\mathcal{NP}$-complete problem class, and is described as $\mathcal{NP}$-hard. For Combinatorial optimisation problems, the usual form of the decision problem is "does there exist a solution to the optimisation problem with a cost less than x". This decision problem trivially reduces to the optimisation problem, and can be shown to be $\mathcal{NP}$-complete.

2.3.2 Phase Transitions

Statistical physics has many similarities to Combinatorial Optimisation that make its techniques and concepts relevant. One of the most important concepts is that of the 'phase transition'. It is well known that many $\mathcal{NP}$-hard problems are rarely hard; for many instances, heuristic algorithms can often find solutions extremely quickly. The idea of phase transitions has helped explain where the difficult instances of $\mathcal{NP}$-complete problems are.

In physical systems, a phase transition is an abrupt, discontinuous change in some property with a small change in some controlling variable, typically the temperature. The classical examples are the transitions between solids, liquids and gases and the transition between ferromagnetic (fixed magnetic alignment) and paramagnetic (magnetic alignment dominated by external magnetic field) phases of magnetic materials. These transitions are between different types of symmetry within the material; the term symmetry breaking is often used in relation to phase transitions.

In Combinatorial Optimisation, phase transitions have been observed in many different problems. The key characteristic that changes between phases is the average difficulty of randomly chosen instances. The difficulty is measured as the average number of iterations of a good performing exact algorithm (e.g. branch and bound) to solve a random instance. For SAT and Max-SAT the control variable is the number of clauses per variable (Mitchell et al., 1992a; Kirkpatrick and Selman, 1994; Monasson et al., 1999). For the Asymmetric Travelling Salesman problem, the critical control variable is the precision of intercity distances, typically represented by the number of digits for the distances (Zhang and Korf, 1996).

The characteristics of the phase transitions differ between decision problems and their corresponding optimisation problem. In decision problems, the transition is described

as "Easy-Hard-Easy". Below the transition, almost every instance has one solution ('yes' or 'no'), and above the transition, almost every instance has the other solution. At or very close to the phase transition, however, almost an equal proportion of instances have each solution. For example, in SAT, almost every instance below the phase transition is satisfiable, while above the transition almost every instance is unsatisfiable. Instances from these two phases are also easy to solve algorithmically. Below the transition there are many satisfying configurations, making it easy to prove that the problem is satisfiable. Above the bound, a contradicting set of clauses can quickly be found, proving the instance is unsatisfiable. Neither of these techniques is effective close to the transition value. Similar quick methods can be found for the easy phases of other problem classes.

In Combinatorial Optimisation, the phase transition tends to be "easy-hard" instead of "easy-hard-easy". The first easy phase in both tends to correspond to finding some limiting minimal state, for example, in Max-SAT, showing that every clause can be satisfied, so there are zero unsatisfied clauses. This is the same for both the decision and optimisation versions of the problem. Above the phase transition, however, the problems become different. In the decision problem, it is only necessary to show that it is not possible to satisfy every clause. For the optimisation problem this is not sufficient; it is necessary to find the maximum number of clauses that can be satisfied, which does not become easier beyond the phase transition.

It is believed that the phase transition is a defining characteristic of $\mathcal{NP}$-hard problems, and it is the phase transition that makes these problem classes difficult. This is difficult to prove in the same manner that it is difficult to prove that $\mathcal{NP} \neq \mathcal{P}$; it is always possible that some undiscovered algorithm could solve instances of $\mathcal{NP}$-hard problems at the centre of the phase transition. There is no way to prove that such a algorithm does not exist. However, the wide range of problems in which phase transition behaviour has been found and the wide range of characteristics that change over the phase transition have led researchers in recent years to identify phase transitions as being the cause of the computational difficulty of $\mathcal{NP}$-hard problems (Zhang and Korf, 1996; Stadler et al., 2003; Kirkpatrick and Selman, 1994; Monasson et al., 1999).

2.3.3 Some examples of $\mathcal{NP}$-Hard Combinatorial Optimisation Problems

Max-SAT

Max-SAT is the optimisation version of the SAT problem. An instance is defined by a set of clauses, where each clause is a boolean formula in conjunctive-normal-form (CNF). The SAT problem is to determine whether any assignment of variables satisfies every clause. The Max-SAT problem is find the assignment of variables the maximises the number of satisfied clauses; we consider the corresponding minimisation problem to be minimising the number of unsatisfied clauses. Often the number of terms in each clause is fixed at some number, and these problems are described as Max-n-SAT, where n is the number of terms in each clause. Note that only Max-n-SAT problems where $n \geq 3$ are $\mathcal{NP}$-complete, as deterministic polynomial time algorithms exist for Max-2-SAT and Max-1-SAT is trivial.

For example a set of 4 clauses with 4 variables might be

$$\{x_1 \vee \neg x_2 \vee x_3,\ x_1 \vee \neg x_2 \vee x_4,\ x_1 \vee x_3 \vee \neg x_4,\ x_2 \vee x_3 \vee \neg x_4\}.$$

If the problem is defined as each clause being a predicate in the list $\mathcal{P}$, the cost function can be written as

$$f(\boldsymbol{x}) = \sum_{p \in \mathcal{P}} [p(\boldsymbol{x})]_0^1$$

where $[expr]_b^a$ is the indicator function that evaluates to a if $expr$ is true and to b if $expr$ is false.

Sometimes a variation known as Weighted Max-SAT is discussed, where each clause is assigned a weight. The cost is then the sum of the weights of the unsatisfied clauses. A Weighted Max-SAT problem can be converted into a normal Max-SAT problem by duplicating clauses so the weight is represented by the number of times a clause is repeated. This is trivial if the weights are integer valued. Rational-valued weights can be converted to integer equivalents by calculating the lowest common denominator of all the weights.

As discussed in section 2.3.2, SAT exhibits a phase transition in the difficult of randomly generated instances as the ratio of clauses to variables, $\alpha = C/N$ is changed. For 3-SAT, this transition is around 4.3.

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is one of the classic 'hard' computing problems. Early versions of the problem were studied in the 1800s. The modern, general case problem, can be stated as finding the minimum cost tour on some weighted graph. An instance is thus specified by a weighted graph. More commonly, the problems are restricted to 2-dimensional maps, where the weights between the nodes is the distance between them. Such problems are easier to solve, as the geometry provides information that can be exploited.

Solving ever larger Travelling Salesman Problems is something of a competition. The current record is the tour through all the cities of Sweden, shown in Figure 2.2. This tour contains 24,978 cities, up from the previous record of 15,112 cities set in April 2001. Fifty years ago Dantzig et al. (1954) reported the solving of a 49 city Travelling Salesman Problem; it is perhaps ironic that as the classic example of an exponentially hard problem, the Travelling Salesman Problem has received so much attention that the size of the largest problem solved has grown exponentially.

Spin-glass

Spin-glasses are problems from physics. A spin-glass is a material which has significant magnetic frustration. For our purposes, a spin glass consists of a set of nodes which can have one of two possible orientations, and a set of constraints, each specifying that two nodes' alignment should be the same or opposite. The objective is to find the set of orientations that minimises the number of unsatisfied constraints. Figure 2.3 is an illustration of a spin glass problem. The connectivity between the nodes can be organised in several different fashions. The nodes can be arranged in a 2 or 3 dimensional grid or the connections can be chosen randomly. The spin-glass problems we consider throughout the rest of this thesis are all fully connected; there is a constraint between every pair of nodes.

Binary Perceptron

The Binary Perception (sometimes called the Ising Perceptron) is a binary version of the classic perceptron (figure 2.4) sometimes used in machine learning. In the perceptron, there are a set of patterns $\boldsymbol{x}_k$, where each pattern $\boldsymbol{x}$ is vector of inputs. Each pattern is associated with a class label c_k. In the Binary perceptron, the problem

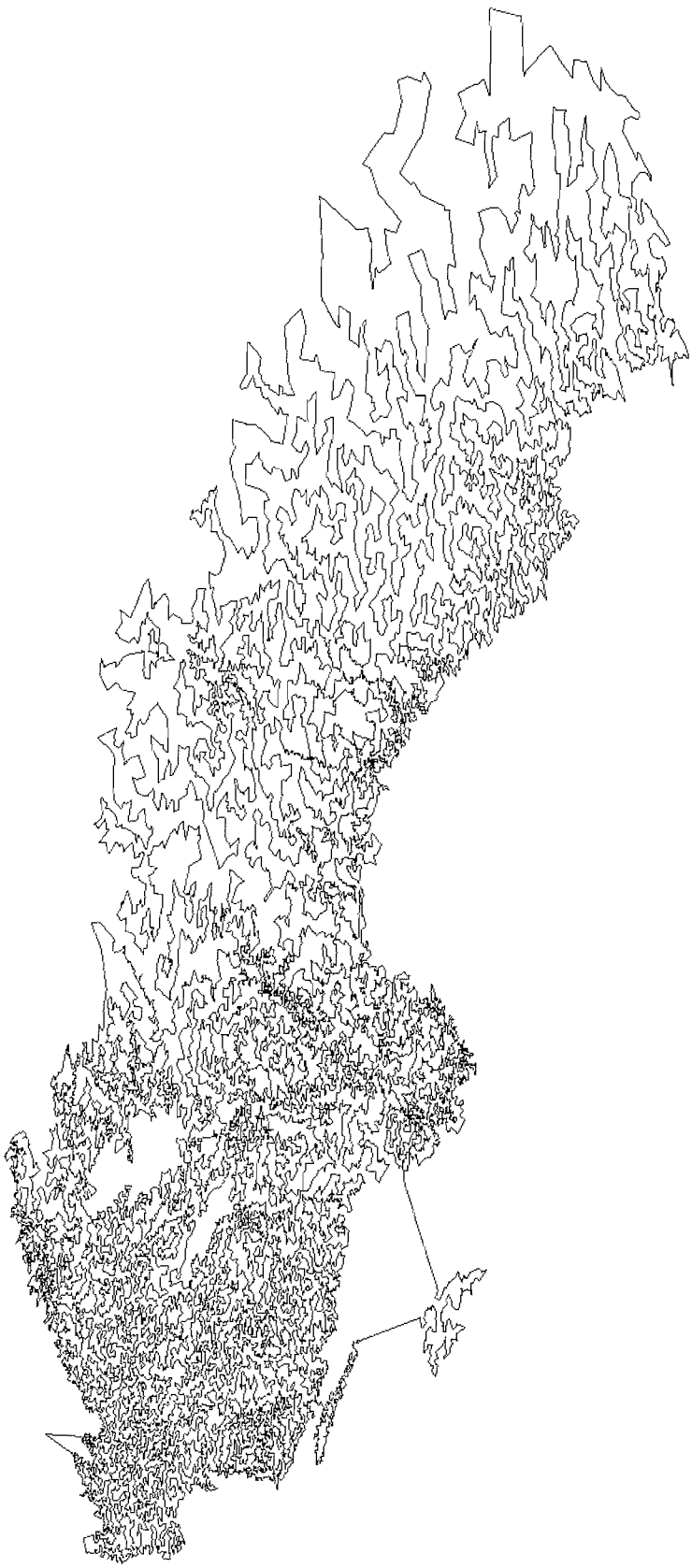


FIGURE 2.2: Outline of the shortest tour through all the 24,978 Cities in Sweden. This was calculated in May 2004 and is currently the largest non-trivial TSP problem that has ever been solved.

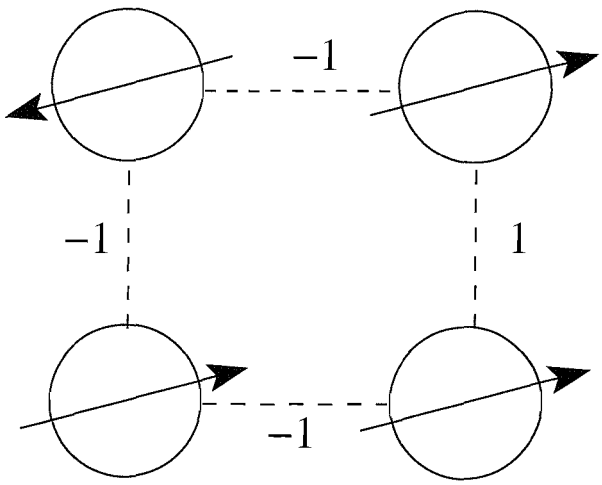


FIGURE 2.3: Illustration of a spin-glass problem. An edge marked with a '1' indicates a constraint requiring the two connected nodes to have the same orientation; edges marked with '-1' indicate a constraint requiring the nodes to have opposite orientations. The constraint on the bottom is unsatisfied with the nodes in their current orientation.

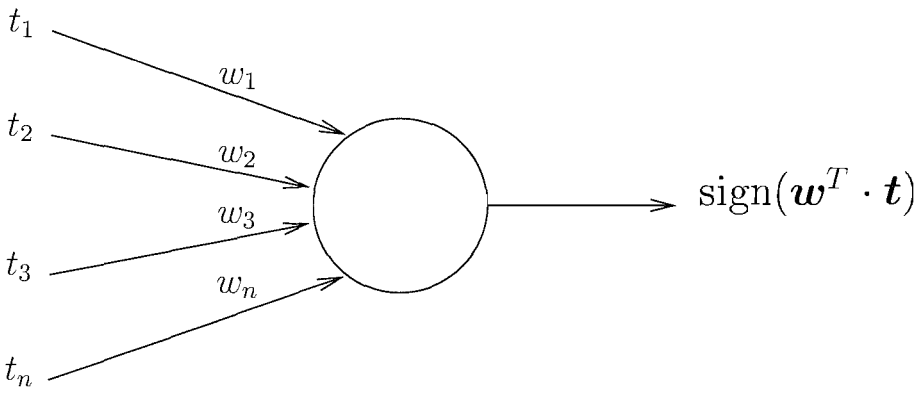


FIGURE 2.4: A Perceptron

is to find the vector of binary weights, $\boldsymbol{w} \in \{-1, 1\}^n$, to minimise the cost function

$$f(\boldsymbol{w}) = \sum_{k=1}^{T} [c_k \boldsymbol{x}_k \cdot \boldsymbol{w} \leq 0]_0^1.$$

The binary perceptron is mainly of theoretical interest, particularly to physicists. In particular, the phase transition in the difficulty of the decision problem version of the binary perceptron has been closely studied as a form of replica symmetry breaking (Prügel-Bennett, 2004a; Krauth and Mézard, 1989).

Chapter 3

Heuristic Search Algorithms And Cost Landscapes

Solving difficult Combinatorial optimisation problems is a major problem in a huge variety of real world situations. Heuristic search algorithms are a vital tool in solving these problems. They are easy to understand, can be applied widely, require little problem-specific modification and often achieve acceptable performance. However, despite much research, heuristic search is poorly understood. There is no concrete theory to guide the design of search heuristics, and only limited understanding of how to predict the performance of differing algorithms. The massive variety and complexity of search algorithms and Combinatorial optimisation problems has presented a barrier to analysis that has not been overcome.

However, several key concepts and ideas have been developed that are useful in describing and understanding Heuristic search algorithms' behaviour, and in comparing behaviour on different problem instances and classes. Of particular importance to this thesis are the ideas of a Neighbourhood and of a Cost Landscape. This chapter describes these concepts and provides the definitions used throughout the rest of this thesis.

This chapter begins with a brief description of Heuristic search algorithms, giving some examples. This is followed by a definition of the idea of Neighbourhood, and a discussion of some of the issues associated with them. Cost landscapes are then defined, and this chapter is concluded with a discussion of several key features of cost landscapes.

3.1 Heuristic Search

Heuristic search algorithms attempt to solve optimisation problems through a process of iterative improvement. In their simpler forms, a Heuristic search algorithm starts by a single possible solution being chosen at random, and set as the current solution. Then at each iteration, a solution that is similar to the current solution is chosen, and its cost is evaluated, and if that cost meets some criteria relative to the current solutions cost, the new solution becomes the current solution. This is continued until some ending criterion is met, such as a certain number of iterations without improvement have elapsed.

This simple outline does not include Genetic algorithms, a more complex form of search algorithm that uses a population of current solutions instead of just a single solution. The process is essentially the same however, and the only difference for our simple outline is that instead of changing the current solution to a new one, the algorithm selects new solutions to add to current solution set, and some to remove from it. A general outline of a heuristic search algorithm is given in listing 3.1

```
1 X ← make_random_solutions();
2 C_X ← getCosts(X);
3 repeat
4     Y ← get_similar_solutions(X);
5     C_Y ← getCosts(Y);
6     (X, C_X) ← choose_new(X, C_X, Y, C_Y);
7
8 until finished()
```

LISTING 3.1: Pseudo code giving an outline of a heuristic search algorithm

$\mathcal{X}$ is the current solution set. The termination condition is normally considered a separate issue to the performance of the algorithm, and from this view, heuristic search algorithms have two key components:

- A similarity operator, used in line 4, to generate new candidate solutions. This is often known as a Neighbourhood operator, but more complex operators such as crossover perform the same task. This operator is probably the biggest influence on a search algorithm's performance, and is discussed in more detail in section 3.1.2.
- Some method of choosing which similar solutions should become part of the solution set, represented by the procedure `choose_new` called on line 7. In the simplest case this simply chooses the lowest cost solutions to become members

of the solution set but more complex methods are also used. This is often known as the *navigation strategy* of the algorithm.

These are the key characteristics of a heuristic search algorithm. Complex algorithms such as Genetic algorithms use several different operators to define each component. Crossover and mutation, for example, are both different ways of generating solutions similar to other solutions, but both can be used by the same algorithm. In these cases it can be useful to consider each operator separately (Jones, 1995).

3.1.1 Example Heuristic Search Algorithms

This section gives some examples of Heuristic Search Algorithms. Since the neighbourhood used by a heuristic search algorithm's is highly problem dependent, this is mainly a comparison of different navigation strategies.

Simple Descent

Simple descent, some times known as greedy search or hill climbing is the most basic form of heuristic search algorithm. A single solution is chosen at random, and new candidate solutions are accepted if they have an equal or lower cost. While simple, this form of descent can be extremely effective, and many problems can be solved using simple descent without resorting to more complicated algorithms.

There is one major problem with simple descent. As the acceptance rule is simple, descent algorithms can easily get trapped in solutions with no lower cost neighbours, and these solutions can often be of a poor quality. These are known as local minima, and are discussed in more detail in section 3.2.1. The simplest way of overcoming this problem is simply to run several descents (starting with a randomly selected solution each time), to increase the chance that local minima are avoided. In addition to this simple method, there are many heuristic search algorithms that can be seen as modifications of descent designed to overcome the problems caused by local minima, such as simulated annealing (discussed in this section) and Tabu search (Glover, 1989, 1990).

Another problem with simple descent is that, unmodified, a long time can be spent revisiting solutions that have already been explored, when a large number of potential solutions have an identical cost. This is often known as a plateau, and is described in more detail in section 3.2.1. This problem has some effect on all heuristic search algorithms, since without differences in cost, an algorithm has no information to guide the search. However, some techniques can lessen the problem. An obvious

method is to cache the costs of visited configurations, so that even if configurations are revisited, little time is wasted in doing so. For large problems, this cache can quickly become impracticably large. One of the motivations behind Tabu search is to improve performance on plateaus.

Simulated Annealing

Simulated Annealing is a variation of simple descent inspired by statistical mechanics. Annealing is the process by which metals or glass are heated to high temperatures to release internal stresses caused by flaws in the microscopic structure. It is necessary to raise the temperature to a point that is high enough for the flaws to be eased, but not so hot for that the item deforms. The glass or metal is then very slowly cooled, so that differences in temperature cannot create new flaws in the material, until a critical temperature is reached, below which flaws cannot form.

In simulated annealing, first proposed in Kirkpatrick et al. (1983) this process is mimicked. Instead of rejecting higher cost solutions outright, higher cost solutions are accepted depending on the difference in cost and a "temperature" T. As in the physical process, the temperature starts high and is reduced as the algorithm continues. The probability of acceptance of a move to a higher cost state is normally calculated by the equation

$$e^{-\Delta C/T}$$

Note that at a temperature of 0, the probability of accepting a higher cost solution becomes 0 and simulated annealing behaves identically to simple descent. Different methods of calculating this probability can be used, but in general the the following properties are expected:

- Moves to equal or lower costs states should always be accepted.
- When $T = 0$, the probability of accepting higher cost states would be 0.
- The probability of accepting a state should be lower for large ΔC than for a small ΔC.

However, the equation listed above allows several properties of the algorithm to be calculated easily as a Markov chain. For example, assuming the algorithm has been run at a constant T for long enough to reach equilibrium, the probability of being in any particular state x is proportional to the cost of that state,

$$P(x) = \frac{e^{\frac{-f(x)}{T}}}{z}.$$

where $z = \sum_{x \in \mathcal{C}} e^{\frac{-f(x)}{T}}$.

In a practical setting, it is also necessary to define some sort of *annealing schedule*, that is, the way that the temperature T changes as the algorithm runs. The annealing schedule is critical to the performance of the algorithm, and it is not known how best to design annealing schedules.

While simulated annealing does have advantages over simple descent, its theoretical use is also of great interest. There is a huge body of theoretical work on simulated annealing, and it is by far the best understood heuristic search algorithm. However, in a practical setting there is little theory that can be applied, and the difficulty in choosing an annealing schedule is a significant problem.

Genetic Algorithms

Genetic Algorithms are search heuristics based upon evolutionary principles. Popularised by Holland (1975), the term now covers a huge range of differing algorithms, for both combinatorial and real valued problems. Genetic Algorithms can be roughly distinguished by the following characteristics

- A population of candidate solutions,
- Some selection criteria which keep some members of the population and discard others,
- A mutation operator,
- Often (but not always) some kind of recombination operator.

The key feature distinguishing Genetic Algorithms from other algorithms is the use of a population instead of a single candidate solution. This is also why Genetic Algorithms have the potential to be more powerful than other algorithms; the use of a population means that the algorithm has far more data to work with than a normal type of algorithm, and if these data can be exploited a Genetic algorithm should be able to outperform simpler algorithms.

In the terminology of Genetic Algorithms, the navigation strategy is called selection. Selection is the process by which candidate solutions are chosen to be included into the population, and members of the population are removed. There are many different forms of selection; one of the most common is binary tournament, illustrated in figure 3.1. In this form of selection, two members are selected at random, and the one with the lower cost is kept as part of the population, and the one with the

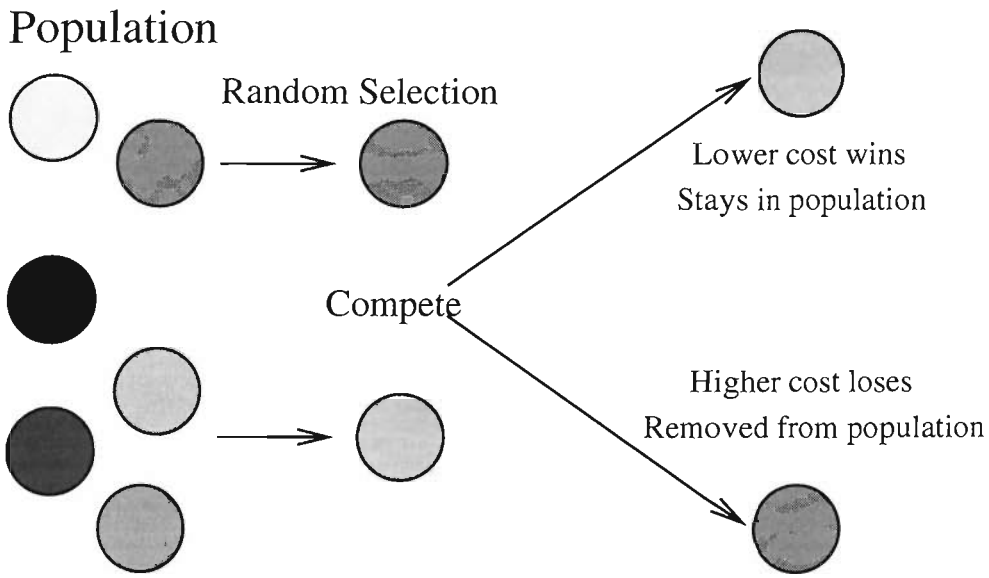


FIGURE 3.1: Binary Tournament

higher cost is discarded. Many other forms of selection have been suggested, such as Boltzmann selection, roulette-wheel selection and stochastic universal selection Baker (1987). Different selection methods vary in the amount of *selection pressure* they apply, how that pressure can be controlled, and stochastic variability of the results of the selection. Selection pressure describes how directed a search is; a high selection pressure causes quicker convergence, at the cost of less exploration and an increased risk of convergence to a poorer quality solution.

Genetic algorithms use a huge variety of methods to generate candidate solutions. Two of the most common are *mutation* and *crossover*. Mutation is an operator that changes candidate solutions by some small amount. Mutation operators are normally fairly simple, for example, flipping bits on a binary string with some small probability. While mutation may slow convergence, it is a simple way of introducing variation into a population, which can be important as certain operators used by Genetic algorithms do not introduce any variation.

Crossover is an operator that creates new candidate solutions from two or more solutions in the population. This can be achieved in many different ways. In 1-point crossover, both parent strings are cut at the same, single point, and then a new solution is generated by joining together the opposite sides of the cut (see figure 3.2). More than one cut can be made, leading to $2, 3$ or n point crossover. These forms of crossover are useful when the variables are ordered in a useful fashion (i.e. variables close together on the string are more closely related than variables far apart). In uniform crossover, at each location in the string, the value for that location is chosen randomly (and independently of any other point) from the values each parent has at

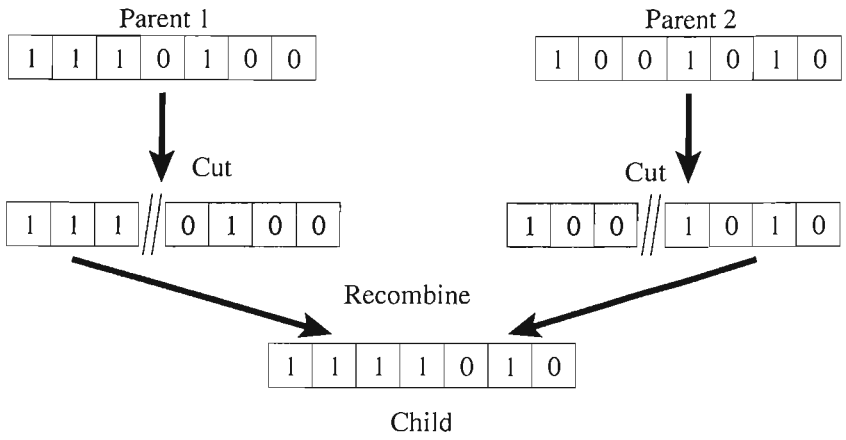


FIGURE 3.2: One-point crossover.

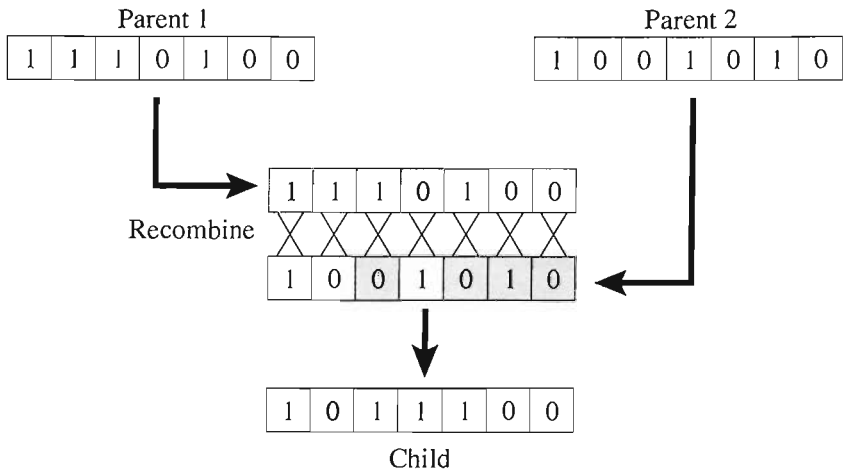


FIGURE 3.3: Uniform crossover.

that location, (see figure 3.3). This form of crossover makes more sense if there is no ordering of the variables.

Genetic algorithms are very challenging to study theoretically. The use of a population creates complex dynamics, and crossover is difficult to study since it is very different to other forms of search heuristic. This lack of a strong theory has contributed to the huge number of different Genetic algorithms that have been devised.

3.1.2 Neighbourhoods

A neighbourhood of a configuration is a set of configurations that are similar to that configuration. Heuristic search algorithms use neighbourhoods to create candidate solutions similar to the best solutions seen so far. The outcome of applying the operators used in a search algorithm to generate neighbours is often represented by

a *Neighbourhood function*, which maps a configuration to the set of configurations in its neighbourhood.

To be useful in a heuristic search algorithm, it must be possible to generate members of the neighbourhood quickly; it is generally assumed that the time taken generating candidate solutions is negligible relative to the time taken evaluating the cost of possible solutions. For this reason, Neighbourhood functions often work directly on the representation (e.g. a binary string, a tour of a graph) of a solution with little reference to any information specific to the problem instance.

The other main criterion of a neighbourhood is that it must produce candidate solutions with a similar cost to the original configuration. This can perhaps be best explained in the context of the *No Free Lunch* theorem (Wolpert and Macready, 1995, 1997). This can be summarised by a quote from Wolpert and Macready (1997)

> ...for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.

An algorithm can only be better than another algorithm on a specific class of problems; in general, every search algorithm is equal. The only way that an algorithm can perform well is for it to be designed for a specific class of problems; over a random class of problems, an algorithm will perform no better than random search.

Heuristic search algorithms can often appear to be so called 'Black-box' optimisers, with the algorithm capable of solving any problem class without any knowledge of the problem class. However, this is not correct; all heuristic search algorithms assume that with the neighbourhood they use, neighbours have a greater similarity in cost, on average, than configurations chosen at random. The neighbourhood function of an algorithm contains the *a priori* knowledge of the problem class, necessary for the algorithm to outperform random search.

In general, a neighbourhood is dependent upon the algorithm as well as the problem class. The use of a particular neighbourhood with reference only to a particular problem class without reference to a particular algorithm or search operator has been criticised (Jones, 1995). However, in practice a particular Combinatorial optimisation problem class has a natural neighbourhood that is used almost universally. For example, in Max-SAT, it is usual for the Hamming neighbourhood to be used, where the neighbourhood of a configuration is all the configurations that vary by exactly one bit. As the configurations vary by only one bit, only a few clauses will be affected (on average), and the cost difference between a configuration and its neighbour is thus limited. In fact, for almost all problems which use a binary encoding, a Hamming

neighbourhood is an appropriate neighbourhood, since in most problems we would expect the change in cost caused by the change in a single bit to be less than changing several bits.

While it is possible to use a different neighbourhood to the natural one, it is often extremely difficult, and requires a thorough understanding of the problem class. Using a different neighbourhood means you are relying on different *a priori* knowledge of the problem, and it is likely that a completely different navigation strategy would be most successful. For these reasons, we tend to assume a single neighbourhood will be used for each problem class.

It is of course possible to create classes of problems which have a binary encoding where the Hamming neighbourhood is not useful, and for these classes a heuristic search algorithm using a Hamming neighbourhood will perform poorly. Also, even within classes where a particular neighbourhood is useful, it is often possible to create pathological instances where a neighbourhood does not meet the criteria necessary to be useful. For example, in Max-SAT, an instance may have a small set of its binary variables in almost every clause, while the other variables are involved in very few clauses. Changing certain bits would then result in a huge change in cost. In these cases, a heuristic search algorithm using that particular neighbourhood would fail. However, since algorithm performance is only meaningfully measured as an average over an entire class of problems, it is sufficient that a neighbourhood is useful for the large majority of instances.

While the idea of neighbourhood is useful for simple algorithms, it can be too simple for more complex operators. Mutation on a binary string (discussed in 3.1.1), where there is a small probability that each bit may flip, technically has a neighbourhood that includes the entire cost landscape, since any configuration can be created by this operation, even though many configurations are extremely unlikely to be created. A more general concept than that of neighbourhoods is a distance measure, represented as function $d : x \in \mathcal{C}, y \in \mathcal{C} \mapsto \mathbb{R}$. Ideally, a distance metric should have the following properties

- $d(x, y) \geq 0$,
- $d(x, y) = 0 \iff x = y$,
- Symmetry $d(x, y) = d(y, x)$,
- Triangular inequality $d(x, y) \leq d(x, z) + d(z, y)$,

for all $x, y, z \in \mathcal{C}$. For operators which can be described sensibly using a neighbourhood function, we can define the distance metric as the minimum number of moves

to a neighbour necessary to move from x to y. For a Hamming neighbourhood, this is the Hamming distance, the number of bits different between the two configurations. This is also a useful distance metric for the mutation operator, as mutation is unlikely to create configurations which have many different bits, and there is a correspondingly large distance between them.

Genetic algorithms are difficult to fit into a cost landscape framework. The use of a population and operators that use more than one solution to create candidate solutions mean that no simple distance metric can be defined. However, in these cases a simpler metric produces a landscape that is still relevant for Genetic algorithms. For example, the Hamming distance is relevant when examining both mutation and uniform crossover on binary strings; configurations with a large Hamming distance are unlikely to be generated from each other by mutation, or by crossover with any string (Stadler, 1999).

3.2 Cost Landscapes

Cost landscapes are a central concept in search algorithm design. The cost landscape is a metaphor for the solution space an algorithm has to explore, with different landscape features hindering or helping the algorithm towards high quality parts of the landscape. Thus, an algorithm can walk along or fall off ridges, and get trapped in basins and ravines.

A cost landscape is defined as consisting of three parts

1. A set $\mathcal{C}$ of configurations
2. A fitness or cost function $f : \mathcal{C} \rightarrow \mathbb{R}$
3. A distance measure on $\mathcal{C}$, $d(x, y)$, as described above.

Notice that this definition does not specify whether the landscape is discrete or real valued, and landscapes are used (sometimes confusingly) to describe solution spaces of either type of problem.

3.2.1 Common Obstacles In Landscapes

One of the most common uses of landscapes is to describe certain features of problem instances that cause difficulty for search algorithms. In this section we give a brief description of the main features that are commonly described in combinatorial optimisation.

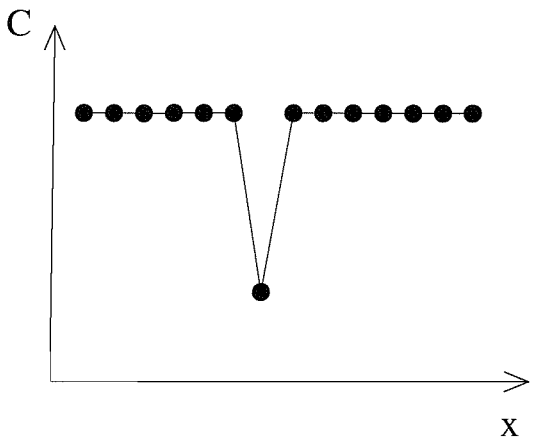


FIGURE 3.4: Golfcourse landscape

Plateau

Plateaus are "flat" areas of landscape, where a collection of connected configurations have equal cost. Plateaus cause big problems for search algorithms, as they can be very large. A major problem with searching plateaus is re-visiting the same configurations many times; a naïve algorithm will random walk over a plateau, and therefore revisit many configurations and take a long time to travel across the whole of the plateau. This problem can be partly overcome by caching costs or by using a Tabu algorithm (Glover, 1989, 1990).

The golf course landscape in figure 3.4 is an extreme example of the difficulties created by plateaus. This is a flat landscape except for the one configuration which forms the global optimum. Beyond using caching or Tabu techniques to avoid re-visiting the same configurations, no search algorithm can expect to do any better than enumerative search, as there is no information to guide the algorithm.

Local Minima

On a continuous landscape, local minima are stationary points with positive second derivatives. On a discrete, non-degenerate landscape (where every configuration has a different cost), local minima are configurations whose neighbours all have a greater cost [1]. Local minima are features which cause difficulty for search algorithms, because to improve the solution once in a local minimum requires that *worse* quality solutions must be accepted.

[1] Defining local minima on degenerate landscapes is not as simple as it may seem; further discussion and *one* definition of local minima on degenerate landscapes is given in section 4.2

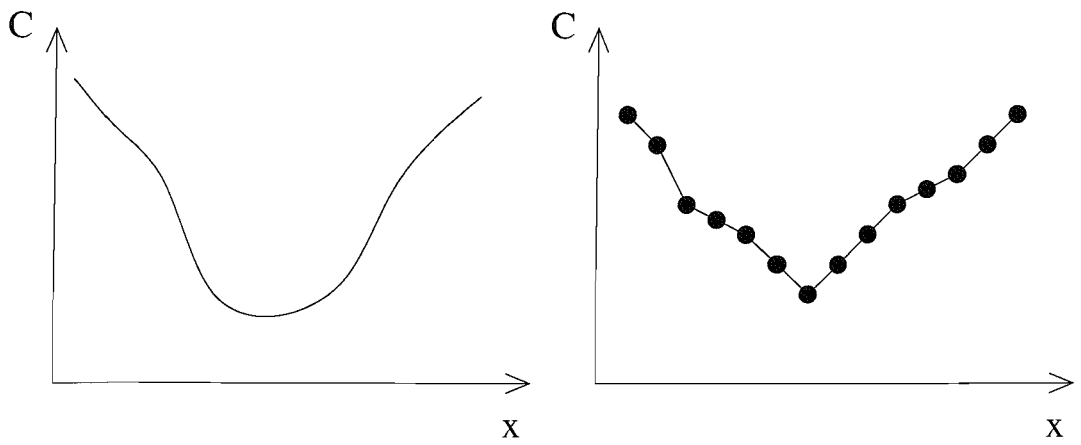


FIGURE 3.5: Example of a local minima on a simple one dimensional continuous (lhs) and discrete (rhs) landscape.

Local minima are one of the main difficulties in search algorithm design. Search algorithms work by focusing the search around the best solutions found so far. Local minima force search algorithm designers to compromise this approach, so that search algorithms can escape from local minima. Algorithms such as simulated annealing and kick-start descent sacrifice convergence time with the hope of escaping low quality local minima in favour of better ones.

Deception

Deception is a property of a landscape which causes search algorithms to be lead away from the global optimum towards a local minima. All landscapes that contain non global local minima can be considered to be somewhat deceptive; figure 3.6 is an example of an extremely deceptive landscape. Note that this landscape is similar to the golf course landscape in that the global optimum has no basin to lead an algorithm towards it. Also note that this deceptive landscape is purely a property of the neighbourhood function used; a different neighbourhood function could create a completely non-deceptive landscape for the same problem instance.

Deception became a topic of great interest in the Genetic Algorithm community after it was identified as a cause of difficulty for Genetic Algorithms (Goldberg, 1987). In terms of the schema theorem (Holland, 1975), deception can be considered as the high fitness small building blocks (or 'low order schema') being contradictory to the high fitness large building blocks (or 'high order schema'). A Genetic Algorithm cannot, therefore, combine high fitness smaller building blocks to make high fitness larger building blocks. Deceptiveness can be considered as local minima for Genetic

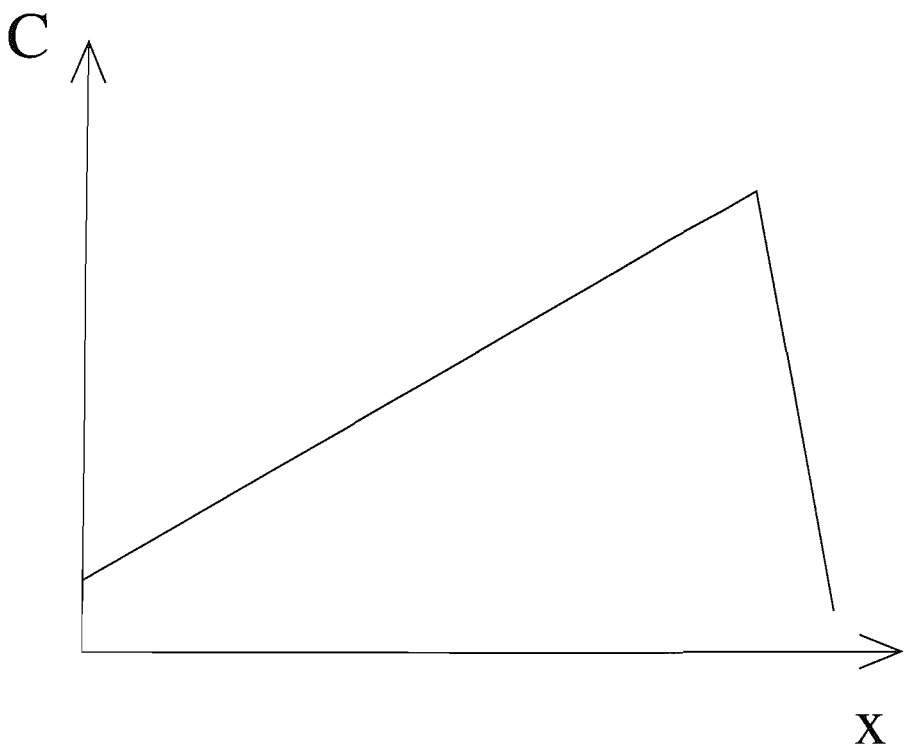


FIGURE 3.6: An extreme deceptive landscape

Algorithms (if an effective landscape could be designed for GAs); a Genetic Algorithm has to move away from the best solutions it has found so far to reach better solutions.

Long Path

The long path problem is a theoretical feature that makes a problem difficult for a search algorithm in the absence of any other feature thought to make landscapes difficult to search, such as local minima or plateaus. In a long path problem, the landscape guides a search algorithm towards a global optimum, but along a long path instead of directly. Therefore, even without local minima or plateaus, a search algorithm can take an extremely long time to reach the global optimum. The long path problem can be visualised best as a maximisation problem, with a hill with the global maximum at the top and the path spiralling around the outside, with a sharp drop in fitness either side of the path stopping an algorithm taking any short cuts. The long path problem is a theoretical problem, and it is not clear how much real algorithms are affected by long path problems as opposed to plateaus and local minima.

Chapter 4

Barrier Trees

Barrier Trees are a tool to study cost landscapes. Cost landscapes are a central concept in search algorithm design, providing a simple, understandable metaphor for algorithm behaviour. But little is known about the characteristics of the landscapes of hard combinatorial problems, and how they affect search. Part of the difficulty is due to the landscape metaphor being misleadingly simple. Features such as saddle-points, slopes and ridges are often thought of in relation to landscapes, but such terms are meaningless in combinatorial landscapes without gradient information. Even relatively simple ideas such as local-minima need to be carefully defined; does a completely flat landscape have no local minima or is every configuration within such a landscape a minima? It is clear that the simple three dimensional landscape, with slopes, hills and hollows implied by the landscape metaphor is misleading, and cost landscapes are actually huge, highly dimensional and have complex topologies.

Barrier Trees are a visualisation technique that attempts to highlight the important features of small cost landscapes. Barrier Trees are well defined, and many of the features of interest to algorithm design have a simple representation on the tree. For example, local minima are leaves on the tree. Barrier Trees also provide a partitioning of the landscape, with every partition representing a node of the tree. This partitioning can be used to analyse the size and shape of particular features, or it can be used to animate a search heuristic. These techniques are demonstrated in chapter 5.

The concepts behind Barrier Trees have developed independently in several different fields, such as protein folding (Becker and Karplus, 1997; Garstecki et al., 1999), RNA secondary structure formation (Flamm et al., 2000) and $\pm J$-spin models (Klotz and Kobe, 1994a,b). Barrier Trees were first used to study combinatorial optimisation problems by Peter Stadler and Christoph Flamm. Their most important work on

the subject is Flamm et al. (2002) which formalises many of the ideas behind Barrier Trees and tackles the difficulty of defining Barrier Trees on a degenerate landscape where several configurations can have an equal cost. In this chapter we present a formalism that is different to that in Flamm et al. (2002). We believe our definition is simpler than that presented by Flamm et al. (2002) and has the advantage of providing a partitioning of the cost landscape in addition to the tree itself. We make considerable use of this partitioning in later chapters.

This chapter provides a formal definition of Barrier Trees including a proof that a Barrier Tree exists for any fully connected landscape. A formal description of the algorithm we use to generate the Barrier Tree is given, and a description of the types of landscapes on which it is practicable to use a Barrier Tree, in particular how Barrier Trees for landscapes that are too large to fully enumerate can be generated. To help with the interpretation of Barrier Trees, a discussion of how several common features of landscapes are represented on a Barrier Tree, and how the Barrier Tree provides a useful formal definition for some of these features is given.

4.1 Definitions

Cost landscapes are defined in chapter 3 as consisting of a set of configurations $\mathcal{C}$, a cost function $f : \mathcal{C} \mapsto \mathbb{R}$ and a Neighbourhood function $\mathcal{N} : \mathcal{C} \mapsto 2^{\mathcal{C}}$. The Neighbourhood functions are related to types of search operators, and the significance of this is discussed in chapter 3. Barrier Trees can be defined on all combinatorial cost landscapes that meet the following criteria:

1. the Neighbourhood function must be symmetric,
2. the landscape must be fully connected.

However, if the landscape is not fully connected, several trees (a *forest*), one for each connected component of the landscape, can be defined instead.

We define a path on the landscape as any connected set, so that the set of all paths is defined as

$$\mathbb{P} = \{\pi \in 2^{\mathcal{C}} | \, (\forall \mathcal{S} \in 2^{\pi} \, \mathcal{S} = \emptyset \vee \mathcal{S} = \pi \vee \, (\exists x \in \mathcal{S} \, \exists y \in \pi \backslash \mathcal{S} \; y \in N(x))) \vee (|\pi| = 1)\}.$$

That is, a path is any set of configurations where every possible subset has a neighbour in the complement of that subset, and single configurations. Strictly speaking, this definition is closer to the concept of a walk, where the same configuration can be

revisited, than a path. This definition is convenient, for example, if we have two paths π_x and π_y, then if $\pi_x \cap \pi_y \neq \emptyset$ then $\pi_x \cup \pi_y$ is also a path under this definition.

Theorem 4.1. *If π_x and π_y are two paths where $\pi_x \cap \pi_y \neq \emptyset$, then $\pi_x \cup \pi_y$ is also a path.*

Proof. If $\pi_x \cup \pi_y$ is a path, then one of the following must be true for all subsets $\mathcal{S} \in 2^{\pi_x \cup \pi_y}$

1. $\mathcal{S} = \emptyset$
2. $\mathcal{S} = \pi_x \cup \pi_y$
3. $\exists x \in \mathcal{S}\; \exists y \in (\pi_x \cup \pi_y) \backslash \mathcal{S}\; y \in N(x)))$

We assume 1 and 2 to be false, so that $\mathcal{S} \neq \emptyset$ and $\mathcal{S} \subset \pi_x \cup \pi_y$. For convenience, we will call the complement of the subset $\mathcal{S}$, $\mathcal{S}\prime = (\pi_x \cup \pi_y)$. $\mathcal{S}$ must intersect π_x or π_y or both. Since $\pi_x \cap \pi_y \neq \emptyset$ and $\mathcal{S} \subset \pi_x \cup \pi_y$, if $\pi_x \subset \mathcal{S}$ then $\pi_y \cap \mathcal{S} \neq \emptyset$ and $\pi_y \not\subset \mathcal{S}$. That is, $\mathcal{S}$ must partially intersect either π_x or in π_y. We take π_z to be the path for which this is true for, so that $\pi_z = \pi_x \vee \pi_z = \pi_y \wedge \pi_z \cap \mathcal{S} \neq \emptyset \wedge \pi_z \not\subseteq \mathcal{S}$

We observe that $\mathcal{S} \cap \pi_z \subset \pi_z$, and as π_z is a path, it follows that there must exist an x and y to satisfy statement 3 above. Since $\mathcal{S}$ was an arbitrary subset and this follows if both statements 1 and 2 are false, one of the statements must be true for all subsets of $\pi_x \cup \pi_y$. It follows that $\pi_x \cup \pi_y$ must be a path. □

A Barrier Tree is a form of Hasse diagram (also known as partially ordered set or poset diagram). A Hasse diagram is a graph representation of a partially ordered set. Our definition of a Barrier Tree therefore consists of first of all defining the members of the set, and then the partial ordering on that set. We then show that the Hasse diagram always takes the form of a rooted tree.

The central concept used to define the members of the set and the partial ordering of the set is *accessibility*. A configuration x is accessible from configuration y at cost c, if there exists a path containing both x and y, and every member of that path has cost less than or equal to c. Formally,

$$\exists \pi \in \mathbb{P}\, (x \in \pi) \wedge (y \in \pi) \wedge (\forall z \in \pi(f(z) \leq c)).$$

Accessibility is a set of relations, indexed by the cost c. We use a special notation, borrowed from Flamm et al. (2002),

$$x \overset{c}{\longleftrightarrow} y.$$

The members of the poset which form the vertices of the Barrier Tree are defined as the equivalence classes of the cost landscape under the equivalence relationship *level-accessibility*. Level-accessibility is defined as a relation,

$$\mathcal{LA} = \{(x, y) | f(x) = f(y) \wedge x \overset{f(x)}{\leftrightarrow} y\}.$$

That is, two configurations are level-accessible if they have equal cost and are accessible to each other at that cost.

Theorem 4.2. *Level-accessibility is reflexive, symmetric and transitive and is therefore an equivalence relation.*

Proof. (i) $f(x) = f(x)$, and the path containing only x satisfies $x \overset{f(x)}{\leftrightarrow} x$ therefore $\mathcal{LA}$ is reflexive.

(ii) We assume x and y are any two configurations for which $(x, y) \in \mathcal{LA}$. This implies $f(x) = f(y)$, and therefore the path satisfying $x \overset{f(x)}{\leftrightarrow} y$ also satisfies $y \overset{f(y)}{\leftrightarrow} x$. Therefore $\mathcal{LA}$ is symmetric.

(iii) We take x,y and z to be any three configurations for which $(x, y) \in \mathcal{LA}$ and $(y, z) \in \mathcal{LA}$ is true. Trivially, $f(x) = f(y) = f(z)$. There exists two paths, π_{xy} and π_{yz}, from x to y and from y to z respectively with every member of equal or lower cost than $f(x)$. Since y is a member of both paths, $\pi_{xy} \cup \pi_{yz}$ is a path that satisfies $x \overset{f(x)}{\leftrightarrow} z$, and it follows that $(x, z) \in \mathcal{LA}$. As x, y and z were any configurations, $\mathcal{LA}$ is transitive.

$\mathcal{LA}$ is reflexive, symmetric and transitive. This is necessary and sufficient for $\mathcal{LA}$ to be an equivalence relation. □

This equivalence relationship is used to partition the landscape into equivalence classes, where every pair of configurations within an equivalency class satisfy the relationship $\mathcal{LA}$,

$$\mathcal{C}/\mathcal{LA} = \{[x] | x \in \mathcal{C}\}.$$

We call these equivalency classes level-accessible sets, and for convenience we label them $\mathcal{P}_1$ to $\mathcal{P}_n$,

$$\mathcal{C}/\mathcal{LA} = \{\mathcal{P}_i | 1 \leq i \leq n\},$$

where $n = |\mathcal{C}/\mathcal{LA}|$ the number of level-accessible sets. We extend the definition of the cost function $f(x)$ so that it can be applied to level-accessible sets and returns the cost that every configuration with a level-accessible set has.

This partitioning can be seen as splitting the landscape along the principal barriers in the landscape; hence "Barrier" tree. If we consider a continuous two-dimensional

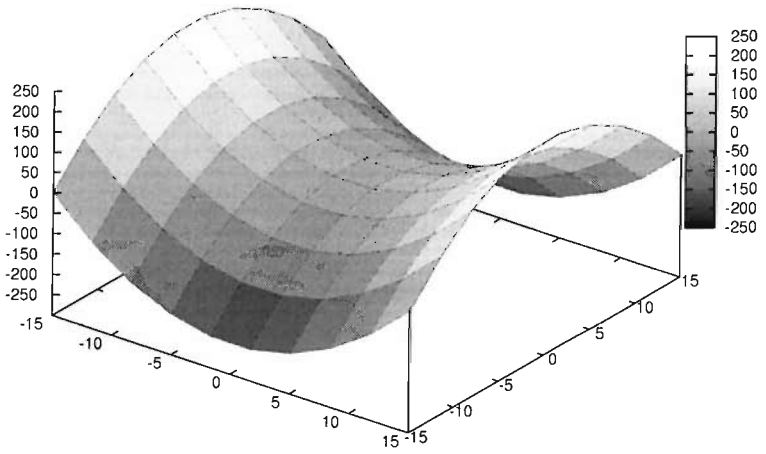


FIGURE 4.1: A saddle of a continuous 2D landscape.

landscape, the barriers are saddle points (see figure 4.1). The barriers in our combinatorial landscape (which each form a level-accessible set themselves) are analogous to these saddle-points, and are referred to as saddle points, although in combinatorial landscapes without gradients the metaphor of a saddle shape is not meaningful.

To complete the poset, a relationship that forms a partial ordering on the set is required. We define a relation between level-accessible sets, $\mathcal{A}$,

$$\mathcal{A} = \{(\mathcal{P}_i, \mathcal{P}_j) | \forall x \in \mathcal{P}_i \ \forall y \in \mathcal{P}_j \ x \overset{f(x)}{\leftrightarrow} y\}.$$

In writing, we say that $\mathcal{P}_j$ is accessible from $\mathcal{P}_i$, without stating at what cost; it is implied that they are accessible at the cost of $\mathcal{P}_i$.

Theorem 4.3. *The relationship $\mathcal{A}$ is reflexive, transitive and anti-symmetric, and therefore defines a partial ordering on $\mathcal{C}/\mathcal{LA}$.*

Proof. (i) From the definition of the accessible sets $\mathcal{P}_i$, $\mathcal{A}$ is trivially reflexive.

(ii)We take $\mathcal{P}_i$,$\mathcal{P}_j$ and $\mathcal{P}_k$ to be any three level-accessible sets for which $(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{A} \wedge (\mathcal{P}_j, \mathcal{P}_k) \in \mathcal{A}$ is true. This implies $f(\mathcal{P}_i) \geq f(\mathcal{P}_j) \geq f(\mathcal{P}_k)$. It follows that there exists two paths π_{ij} and π_{jk}, which contain all the configurations in $\mathcal{P}_i$, $\mathcal{P}_j$ and $\mathcal{P}_j$,$\mathcal{P}_k$ respectively, and every configuration in these paths has cost less than or equal to $\mathcal{P}_i$. Therefore every member of the path $\pi_{ij} \cup \pi_{jk}$ has a cost lower than $\mathcal{P}_i$ and connects $\mathcal{P}_i$ to $\mathcal{P}_k$, which is necessary and sufficient for $(\mathcal{P}_i, \mathcal{P}_k) \in \mathcal{A}$. Since $\mathcal{P}_i$,$\mathcal{P}_j$ and $\mathcal{P}_k$ were any level-accessible sets, $\mathcal{A}$ is transitive.

(iii) We take $\mathcal{P}_i$ and $\mathcal{P}_j$ to be two level-accessible sets for which $(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{A}$ and $(\mathcal{P}_j, \mathcal{P}_i) \in \mathcal{A}$. Trivially, $f(\mathcal{P}_i) = f(\mathcal{P}_j)$. By definition, $\forall x \in \mathcal{P}_i \; \forall y \in \mathcal{P}_j \; x \overset{f(x)}{\leftrightarrow} y \wedge$ $f(x) = f(y)$, and it follows that $\forall x \in \mathcal{P}_i \; \forall y \in \mathcal{P}_j \; (x, y) \in \mathcal{LA}$. Since $\mathcal{P}$ is defined as the partitioning under $\mathcal{LA}$, $\mathcal{P}_i = \mathcal{P}_j$. Since $\mathcal{P}_i$ and $\mathcal{P}_j$ were any level-accessible set, $\mathcal{A}$ is anti-symmetric.

$\mathcal{A}$ is therefore reflexive, transitive and anti-symmetric, so forms a partial ordering on $\mathcal{P}$. □

In partial ordering terminology, if $(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{A}$ then $\mathcal{P}_j$ is a successor of $\mathcal{P}_i$ and $\mathcal{P}_i$ is a predecessor of $\mathcal{P}_j$. This partial ordering completes the definition of the poset that forms a Barrier Tree. A Hasse diagram is a graph of a poset, where the nodes of the graph are members of the set, and edges indicate *covering relations.* A cover relation is a reduction of a partial order with only the *immediate* successor included. For an arbitrary partial order $\mathcal{O}$, the cover relation $\mathcal{O}\prime$ is

$$\mathcal{O}\prime = \{(x, y) \in \mathcal{O} | \nexists z \; z \notin \{x, y\} (x, z) \in \mathcal{O} \wedge (z, y) \in \mathcal{O}\}$$

For our accessibility relationship this is:

$$\mathcal{A}\prime = \{(\mathcal{P}_j, \mathcal{P}_i) \in \mathcal{A} | \nexists \mathcal{P}_k \, \mathcal{P}_k \notin \{\mathcal{P}_i, \mathcal{P}_j\} \, (\mathcal{P}_j, \mathcal{P}_k) \in \mathcal{A} \wedge (\mathcal{P}_k, \mathcal{P}_i) \in \mathcal{A}\}.$$

This can be read as 'there is an edge between $\mathcal{P}_j$ and $\mathcal{P}_i$ if $\mathcal{P}_j$ is accessible from $\mathcal{P}_i$ and $\mathcal{P}_j$ is not accessible from any other member of $\mathcal{C}/\mathcal{LA}$ that is accessible from $\mathcal{P}_i$'.

In summary the Barrier Tree can be defined, given the equivalence relationship $\mathcal{LA}$ and the partial ordering $\mathcal{A}$, as a graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$

$$\begin{aligned} \mathcal{V} &= \mathcal{C}/\mathcal{LA}, \\ \mathcal{E} &= \{(\mathcal{P}_i, \mathcal{P}_j) | (\mathcal{P}_j, \mathcal{P}_i) \in \mathcal{A} \wedge \nexists \mathcal{P}_k ((\mathcal{P}_j, \mathcal{P}_k) \in \mathcal{A} \wedge (\mathcal{P}_k, \mathcal{P}_i) \in \mathcal{A})\}. \end{aligned}$$

A Hasse diagram forms a connected tree if it contains one member that is a predecessor to every other member, and every member can be shown to be covered by at most one member.

Theorem 4.4. *The Hasse diagram defined by the partially ordered set* $(\mathcal{C}/\mathcal{LA}, \mathcal{A})$ *is a connected tree on a fully connected set of configurations.*

Proof. (i) Given that the cost landscape is finite, there exists some level-accessible set, $\mathcal{P}_{MAX} \in \mathcal{V}$, with the maximum cost value of any configuration in the landscape. The cost landscape is fully connected, so a path must exist from $\mathcal{P}_{MAX}$ to every

other level-accessible set in $\mathcal{V}$. Since every configuration has a cost lower than or equal to $\mathcal{P}_{MAX}$, every other level-accessible set is accessible from it, and $\mathcal{P}_{MAX}$ is a predecessor to every other member.

(ii) A Hasse diagram is acyclic if every element is covered by at most one member. Assume a level-accessible set $\mathcal{P}_x$ is covered by two other level-accessible sets, $\mathcal{P}_a$ and $\mathcal{P}_b$, such that $f(\mathcal{P}_a) \geq f(\mathcal{P}_b)$ and $\mathcal{P}_a \neq \mathcal{P}_b$. Since $\mathcal{P}_x$ is accessible from both $\mathcal{P}_a$ and $\mathcal{P}_b$ at the cost of $f(\mathcal{P}_a)$, and accessibility is symmetric and transitive, $\mathcal{P}_b$ is accessible from $\mathcal{P}_a$. This is a contradiction of the statement that $\mathcal{P}_a$ covers $\mathcal{P}_x$ or the statement the $\mathcal{P}_a \neq \mathcal{P}_b$, therefore a level-accessible set cannot be covered by two other level-accessible sets.

Therefore, the Barrier Tree is a rooted tree. □

An example Barrier Tree is shown in Figure 4.2. We only draw from the highest cost saddle point downwards; there are level-accessible sets at every cost level above the displayed root of the tree, however they are structurally uninteresting, each node having only one child each. Each merging node is drawn as a horizontal line to improve the clarity of the tree. The vertical axis is used to indicate the cost of each level-accessible set and is labelled C. Only the leaves and merging nodes are represented on the tree, nodes with only one child are not marked. For example, in figure 4.2, there are six level-accessible sets with cost 2. The level connected set with cost 2 attached to the global minimum is not explicitly shown.

4.1.1 Level-Accessible and Level-Connected Sets

Level-accessibility is used as a partitioning as it forms a tree when accessibility is used as a partial ordering. An alternative, and perhaps simpler partitioning to level-accessible sets is what we call level-connected sets. Two configurations are in the same level-connected set if there exists a path between them where every member of that path has equal cost. A level-connected set corresponds to the idea of a plateau on the landscape. Level-connectedness is a more restrictive relation than level-accessibility, so a level-accessible set is made up of one or more level-connected sets. At first appearance, level-connected sets may seem to be a more natural partitioning for handling degenerate landscapes. However, accessibility is not a partial ordering on level-connected sets, as it is not anti-symmetric. It is still possible to create a graph like structure from level-connected sets, however this forms a merging graph (Flamm et al., 2002) instead of a tree. Merging graphs are more complex to visualise and understand, and are therefore considerable less useful than a Barrier Tree. The other limitation of level-connected sets is that the number of level-connected sets

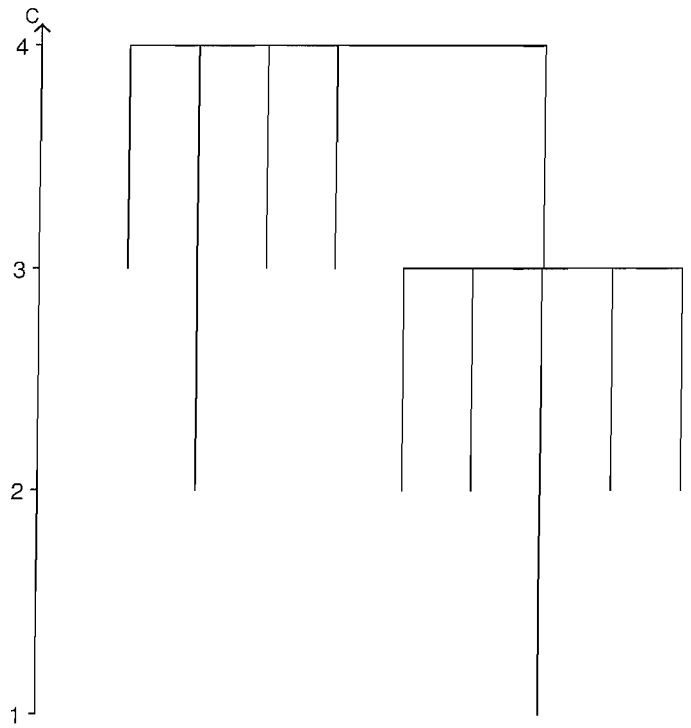


FIGURE 4.2: Example of a Barrier Tree. Landscape represented is of a 20 variable MAX-3-SAT problem with 100 clauses.

that partition a landscape can be orders of magnitude larger than the number of level-accessible sets, severely limiting their use as a visualisation and analysis tool. An extreme example of this is the Ones-counting problem where the cost of a binary string is the number of '1s' in that string. In this case, if a Hamming neighbourhood is used every configuration is a separate level-connected set, making 2^N level-connected sets, while there are only $N + 1$ level-accessible sets.

We make substantial use of level-connected sets in the Barrier based models described in chapters 6 and 7. In these models, basing them on level-connected sets removes what could be a significant source of error, although the difficulties caused by the increased number of partitions remain.

4.2 Features of a Barrier Tree

The formal definition of a Barrier Tree does not give a very intuitive understanding of what the different features of a Barrier Tree are. In this section, we try to give a better idea of what the different parts of a Barrier Tree represent. Figure 4.3 will be used to illustrate the features being described.

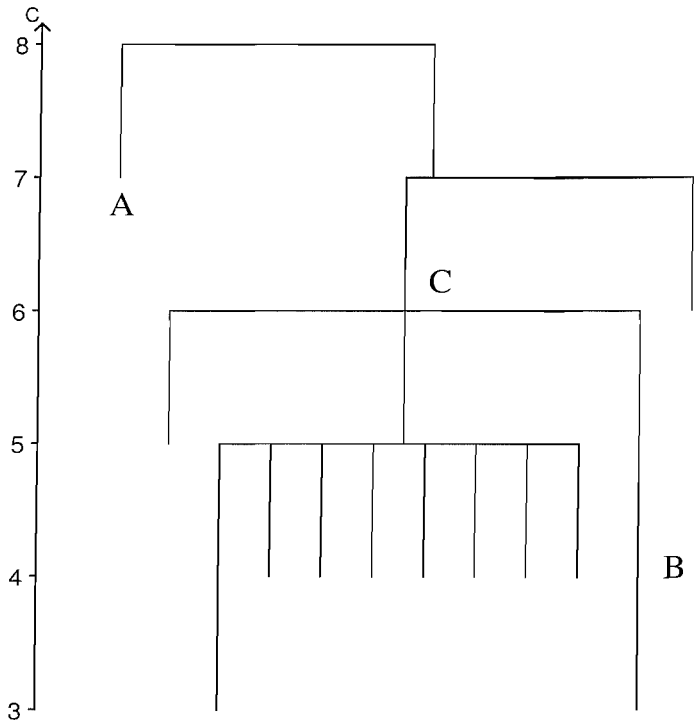


FIGURE 4.3: A Barrier Tree with a local minimum (A), part of a basin (B) and a saddle point (C) marked. Landscape represented is of a MAX-3-SAT problem with 20 variables and 120 clauses.

Leaves

Each leaf on the tree (example marked 'A' on Figure 4.3) represents a local minimum. While local minima are an important concept in optimisation literature, there can be some confusion as to what exactly a local minimum is on a Combinatorial landscape, without gradient information and where several configurations can have the same cost. Definitions which look at only a single configuration tend to have difficulty in distinguishing shoulders from basins (see figure 4.4).

The definition that follows naturally from the definition of Barrier Trees is that a local minimum is a level-accessible set from which no other configurations are accessible. This is illustrated in figure 4.4. Note that for local minima the level-accessible set is made up of only one level-connected set. The definition of whether a configuration is a local minimum or part of a local minimum is now non-local; figure 4.4 shows how a configuration being a part of a local minimum can be decided by the cost of configurations several steps away, not just on a configuration and its neighbours. The advantage of this definition is that it is intuitively closer to what we consider to be a local minimum.

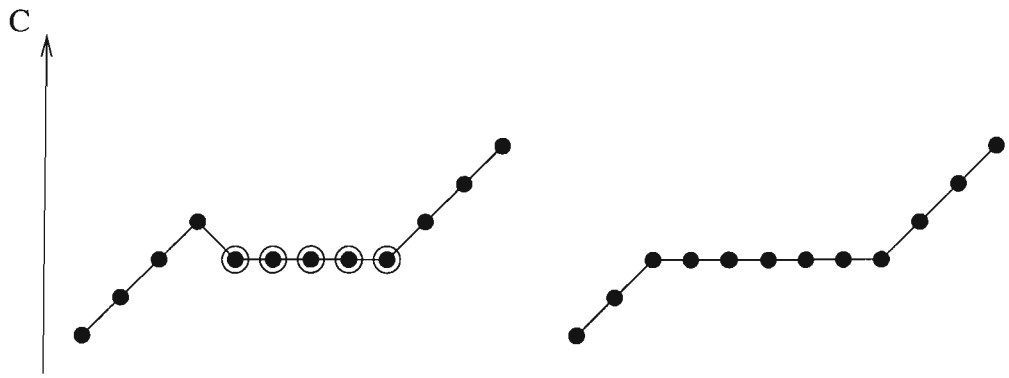


FIGURE 4.4: The circled nodes in the l.h.s. are part of a single local minimum, using a definition based on accessibility. There is no local minimum on the r.h.s. even though only one node has changed.

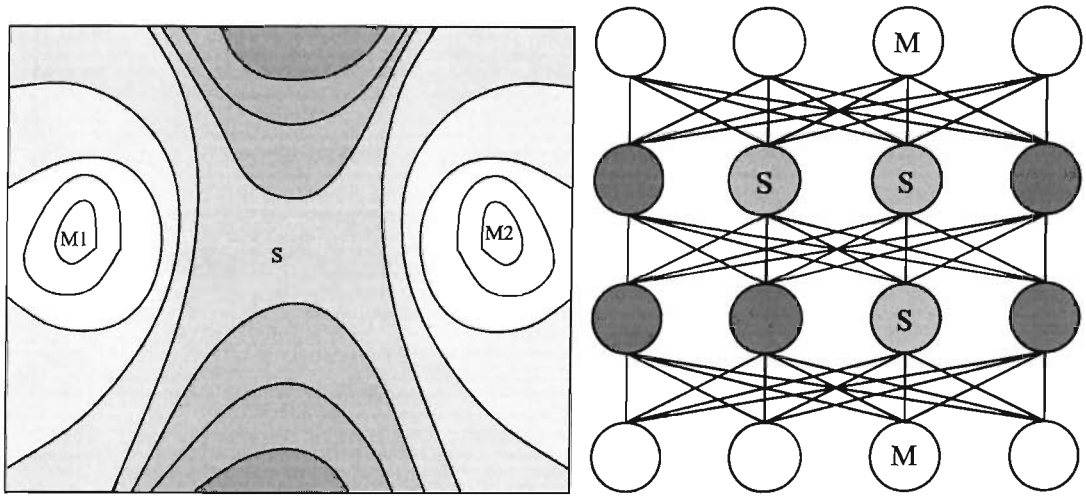


FIGURE 4.5: The l.h.s. shows a contour plot of a saddle-point on a continuous 2D landscape, with lighter shades of grey indicating a lower cost. The r.h.s. shows what we call a saddle-point in a discrete landscape.

Merging Nodes

Merging nodes are internal nodes with more than one child, as marked with a 'C' in figure 4.3. They are the lowest cost accessible sets from which their descendants are accessible. There is some correspondence to the idea of a saddle-point in continuous landscapes, illustrated in figure 4.5. Like saddle-points, merging nodes are the lowest cost point from which two or more minima can be reached by only going "down hill".

However, a merging node may contain level-connected sets from which only one local minimum can be reached. Figure 4.6 illustrates this on a simple landscape. Both local minima can only be reached by moves to lower cost solutions from 'S', however the other two circled solutions are part of the same level-accessible set.

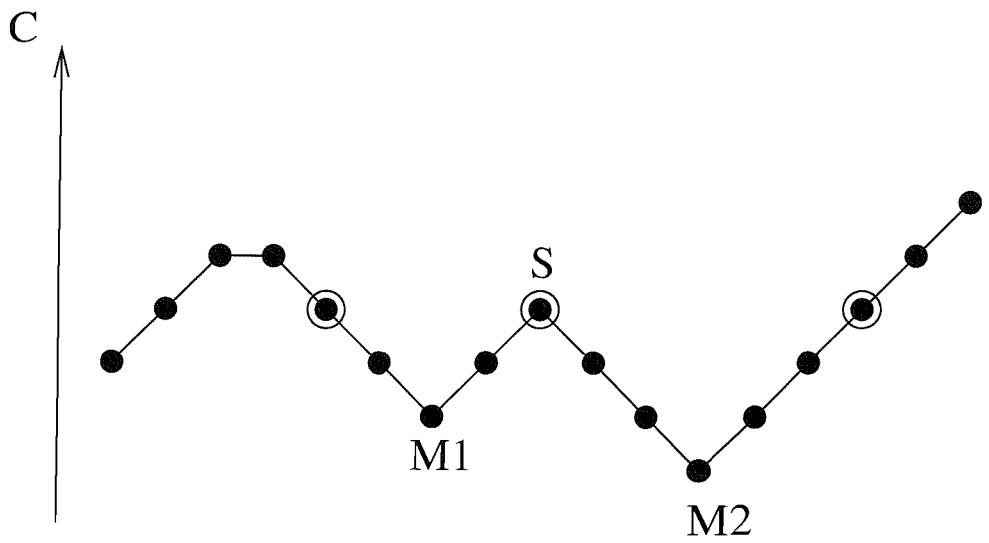


FIGURE 4.6: Simple 1-D landscape demonstrating the difference between a level-accessible set of a merging node on a Barrier Tree and a saddle point. The circled nodes make up a level accessible set which will be the merging node between the local minima 'M1' and 'M2', however only the node marked with 'S' could be considered a saddle point.

Non-merging Nodes

Nodes with only one child, as marked with a 'B' on figure 4.3, do not have any representation in our diagrams, as we do not explicitly draw every node. This has not been an issue with the problem classes we have looked at, since they have integer cost-levels, and typically have a large number of configurations per cost-level, so the chance of a 'gap' in a branch at any particular cost-level is very small.

Non-merging nodes represent level-accessible sets which can be thought of as being parts of a basin around their descendant node. Around a local minimum, a basin represents the configurations from which only the local minimum can be reached if only moves to equal or lower cost states are accepted (for example, a descent algorithm). Above a saddle point, a basin is a less intuitive concept. Barrier trees provide a useful definition that meets one possible concept of a basin around a configuration x

$$B(x) = \{c \in \mathcal{C} | c \underset{\rho}{\overset{f(c)}{\not\rightarrow}} x \wedge \nexists y \in \mathcal{C} \, (c \underset{\rho}{\overset{f(c)}{\not\rightarrow}} y \wedge y \underset{\rho}{\overset{f(y)}{\not\rightarrow}} x)\}.$$

4.3 Larger Landscapes

So far, it has been assumed that the landscape has been fully evaluated. This limits us to very small problems; around 8 million configurations or 23 binary variables. However, we have less interest in the higher cost areas of the landscape, in particular

those above the highest cost barrier in the landscape. Configurations with a cost higher than this barrier can access every configuration with an equal or lower cost. The Barrier Tree representation of this part of the landscape is therefore just a straight line of nodes with a single child each (we automatically cut this part of a Barrier Tree off our diagrams, and use the first splitting node as the root). The Barrier Tree tells us very little about this part of the landscape.

As we have little interest in this part of the landscape, it makes little sense to enumerate the higher cost solutions. We use a modified branch-and-bound algorithm with a fixed bound, to only generate solutions below a fixed cost (Land and Doig, 1960). We attempt to set this cost above the highest cost saddle-point. Unfortunately, while it is possible to calculate certain expected statistics on the cost landscape (Reeves and Eremeev, 2004) this cannot be guaranteed. If the partially evaluated landscape is not fully connected, that gives us a clear indication that there is a saddle point at a higher cost than our bound. The resulting Barrier Tree will actually be a forest (i.e. several trees). However, a fully connected partial landscape does not guarantee that the highest cost saddle point has been evaluated, because there may be a local minimum at a higher cost than we have evaluated. In general, there is no way that we can guarantee there are no local minima above a particular cost without evaluating the entire landscape. To gain some confidence in the tree evaluated from a partial landscape, the end results from multiple descents be can examined. If all descents end at a cost below our threshold, we can assume that any missed local minima have very small basins of attraction and are therefore of little importance to search.

The algorithm used to generate Barrier Trees described in the next section, is correct up to any particular cost level. That is, the algorithm does not require higher cost configurations to generate the Barrier Tree for lower cost configurations. If the lower cost configurations given to the algorithm are not connected, then the algorithm will output a set of trees, one for each disconnected part of the landscape.

The use of a branch and bound method unfortunately introduces a problem specific algorithm. A branch and bound technique can be applied to any problem where a bound can be calculated on the best possible solution containing a partial solution. The efficiency of a branch and bound technique depends on how tight the bound is and on the heuristic used to choose how to build the partial solution. Branch and bound techniques have been developed for a wide range of problems including spin-glass, MAX-SAT and the Binary perceptron problem. However, we have only implemented the technique on MAX-SAT, using the Davis-Putnam-Loveland algorithm as a basis (Borchers and Furman, 1999; Davis and Putnam, 1960; Davis et al., 1962).

4.4 Barrier Trees Algorithm

Barrier Trees are generated by a 'flooding' style algorithm. The configurations are sorted into order of cost, and the low cost configurations are processed first. We will call the set of configurations at a particular cost a cost-level, and the algorithm is best understood by examining the algorithm at a single cost level.

As input the algorithm takes a cost landscape as defined in chapter 3, made up of a set of configurations $\mathcal{C}$, a cost function $f : \mathcal{C} \rightarrow \mathbb{R}$, and a neighbourhood function $N : \mathcal{C} \rightarrow 2^{\mathcal{C}}$. For convenience we label all configurations with a cost c as $\mathcal{C}_c$. As output, the algorithm produces three main structures:

1. A mapping, M, from every configuration to its level-connected set. Each level-connected set is represented by a unique identifier
2. A mapping, A, from every level-connected set to the level-accessible set of which it is a part.
3. A tree structure of level-accessible sets, $\mathcal{T}$. When the algorithm is complete this is the Barrier Tree; as the algorithm progresses this is the set of subtrees of the Barrier Tree below the current cost level.

In this section, M and A are taken to be mappings to sets; a set of configurations in a level-connected set for M, and a set of level-connected sets in a level-accessible set for A. In practice, it is more practicable to represent level-connected sets and level-accessible sets with a unique identifier, rather than passing around set constructs. Therefore, when implementing the algorithm, M and A should be mappings to a unique identifier, and a separate mapping should then point to the sets that make up the level-connected sets and level-accessible sets.

The algorithm can be best understood by considering how a single cost level is processed. The algorithm starts with the lowest cost level and works through each cost level in order until the entire landscape is processed. Each cost level is processed independently, with the exception that it is assumed that M, A and $\mathcal{T}$ have been correctly calculated for the lower cost levels. The top level algorithm is given in listing 4.1. The algorithm first loads the problem landscape and initialises some values. Each cost level is then processed from the lowest to the highest cost.

Most of the processing occurs within the procedure `add_cost_level`. Pseudo code for this procedure is given in listing 4.2. Firstly, the procedure `get_level_connected_sets` calculates the level-connected sets at this cost level, and adds the associated mappings

```
C ← load_landscape()
M ← ∅;
A ← ∅;
T ← ∅;

//enumerate cost levels from low cost to high cost
for Cc=CMIN to CMAX
    (M, A, T) ← process_cost_level(Cc, M, A, T);
endfor
```

LISTING 4.1: Top level of algorithm to generate Barrier Trees

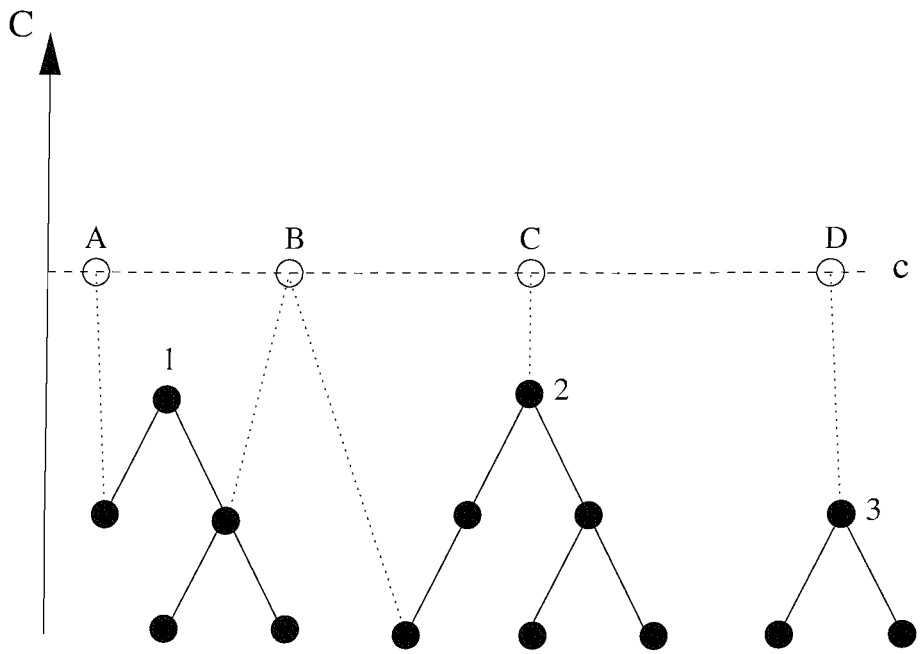


FIGURE 4.7: A partially completed Barrier Tree, just prior to cost level c being processed. The filled in circles represent level-accessible sets, arranged into subtrees in $\mathcal{T}$. The hollow circles represent level-connected sets at the current cost level that have not yet been collated into level-accessible sets. The dotted lines indicate a neighbour connection from a connected set to a level-accessible set. A,B and C are all accessible to each other through lower cost branches so are part of the same level-accessible set.

```
procedure add_cost_level(C_C, M, A, T)
{

   (L, M) ← get_level_connected_sets(C_C, M);

   forall l ∈ L //enumerate through level-connected-sets
      connected_branches ← ∅;
      //going to iterate through every neighbour
      //of every configuration in this level connected-set
      neighbours ← ⋃_{x∈l} N(x);
         forall n ∈neighbours
            if f(n) < C then
               //only interested in lower cost neighbours
               //they have been assigned to a branch in T
               branch ← find_branch(A(M(n)), T));
               connected_branches[l]
                        ← connected_branches[l] ∪ {branch};
            endif
         endfor
    endfor

    (A, A_NEW) ← cluster_LCS(connected_branches, L, A);

    //now go through every new accessible set
    //and add them  to the tree in T
    forall a ∈ A_NEW
       descendant_branches=⋃_{l∈a} connected_branches[l] ;
       if |descendant_branches|=0 then
          //no lower connections, so a local minimum
          T ← add_subtree(T, a);
       else if |neighbour_branches|=1
          // 1 lower connection, so part of basin
          T ← add_to_subtree(T, descendant_branches);
       else if |neighbour_branches| > 1
          //several lower branches, so a merging node
          T ← join_subtrees( T, descendant_branches);
       endif
    endfor
    return (M, A, T);
}
```

LISTING 4.2: Algorithm for adding a single cost level to Barrier Tree

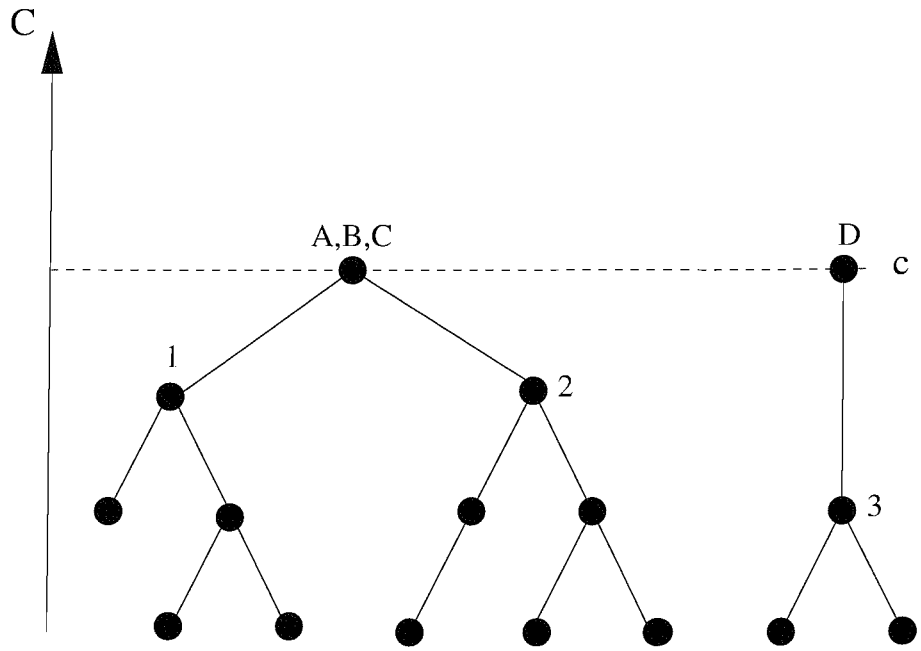


FIGURE 4.8: The same tree as in figure 4.7 after cost level c has been processed.

from configurations to each level-connected set to M. This is a standard clustering algorithm, which we will not detail further here.

Each level-connected set is then examined in turn. The neighbours of every configuration within the level-connected set are found [1]. For each neighbour with a lower cost, the subtree in $\mathcal{T}$ that the neighbour is part of is found. Each subtree is a branch of the Barrier tree that is joined to the other subtrees by merging node with a cost higher than the current cost level. In figure 4.7 the subtrees are labelled 1,2 and 3. The subtree each neighbour is in is found by the procedure `find_branches`. The set of branches in which each level-connected set has neighbours is stored in `connected_branches`, which is a map from a level-connected set to the set of branches to which it is directly connected.

Any two configurations that map to the same branch in $\mathcal{T}$ are accessible to each other at the current cost level. Our mapping `connected_branches` gives us the branches that are trivially accessible from each level-connected set; they are trivially accessible because there is direct neighbour relation between them. Since accessibility is transitive, any two level-connected sets that can access the same branch must also be accessible to each other. The procedure `cluster_LCS` uses this to calculate the level-accessible sets. This is a similar clustering algorithm as used by `get_level_connected_sets`, except it collates level-connected sets that can access the same branches, instead of configurations that neighbour each other.

[1] This can be efficiently calculated at the same time the level-connected sets are calculated by `get_level_connected_sets`.

The procedure `cluster_LCS` adds mappings to A from each level-connected set to its level-accessible set, and returns the level-accessible sets at the current cost level in a set A_{NEW}. All that remains to do is to calculate how to update the tree structure $\mathcal{T}$ for this cost level. For each level-accessible set, all the connected branches of the level-connected sets within the level-accessible set are collected together into `descendant_branches`. If the level-accessible set is connected to no branches, it is a local minimum, and is added to $\mathcal{T}$ by the procedure `add_subtree`. If it is connected to a single branch, it is a part of that branch, and is added to it with the procedure `add_to_subtree`. Finally, if it is connected to several branches, those branches are now connected to each other (i.e. they are accessible to each other at the current cost level though the level-accessible set). The branches are joined by the procedure `join_subtrees`.

Figure 4.7 shows a partially completed Barrier Tree before the level-connected sets at cost c have been collected together into level-accessible sets. Figure 4.8 shows the same tree after the cost level c has been processed.

The algorithm for an individual cost level is correct, as long as the M,A and $\mathcal{T}$ for all lower cost levels have been correctly calculated, so that configurations that are in the same branch in $\mathcal{T}$ are accessible to each other at the current cost level. At the lowest cost level, M, A and $\mathcal{T}$ are all trivially empty, so it is possible to build up a correct Barrier Tree from the lowest cost level up to the highest.

Chapter 5

Barrier Tree Examples

This chapter demonstrates the use of Barrier Trees, both as a visualisation and an analytical tool. It is only a demonstration of Barrier Trees and various techniques that can be used with Barrier Trees; to draw any meaningful conclusions on problem landscapes a large number of instances would need to be studied and analysed, which would require more computational resources than we have available. This chapter aims to demonstrate the wide range of features that Barrier Trees can emphasise and be used to measure.

5.1 Comparison of Different Landscapes

A useful ability of a visualisation tools such as Barrier Trees is the large amount of information that humans can easily interpret in visual form. Barrier trees present the large scale landscape features of a problem in a form that hopefully allows an overview of the landscape to be understood 'at a glance'. Many complex features can be shown by a Barrier Tree, and these features do not have to be explicitly searched for. These features can provide insight into what the important features of a landscape are, and allow researchers to gain an instinctive understanding and characterisation of landscapes. This characterisation is best demonstrated by comparing example Barrier Trees of different landscapes, which allows clearly identifiable visual features to be picked out. This section presents several comparisons of these types, firstly showing very distinct differences between different classes of problems, and then more subtle differences between problem landscapes from the same class of problems.

5.1.1 Different Problem Classes

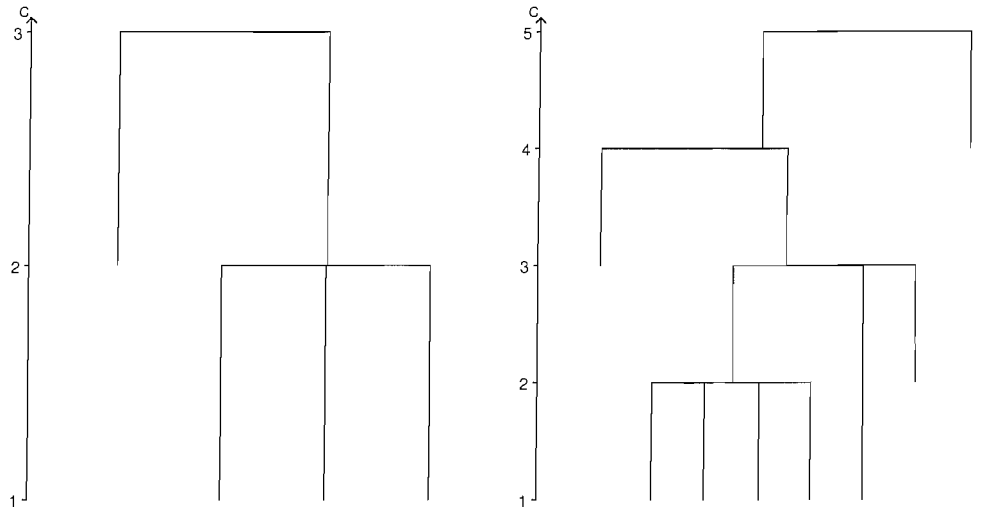
Figure 5.1 shows a comparison of example landscapes from three classes of problems, Max-SAT, Spin-glasses and Binary-perceptron, all described in chapter 2. The Max-3-SAT and Spin-glass instances have 20 variables, and the Binary Perceptron instances have 13; while we can generate Barrier Trees for Binary Perceptron problems with 20 variables, the high number of local minima (around 8000) means they are difficult to draw using our normal tree layout. Where there is a complexity parameter (as described in section 2.3.2), this has been set at the easy-hard transition. Distinctive differences can be seen at a glance. The spin-glass problem is symmetric, with the binary complement of any solution having an identical cost. This symmetry in the landscape is reflected in their trees; the trees aren't actually symmetric, but the fact that every saddle point has an identical partner is immediately clear. The trees of the landscapes of Max-3-SAT problems and Binary Perceptron problems are also distinctive, with the Binary Perceptron landscapes having a high number of leaves spread over relatively few cost levels, while the Max-3-SAT trees have relatively few local minima but more complex subtrees.

5.1.2 Different Instance Sizes

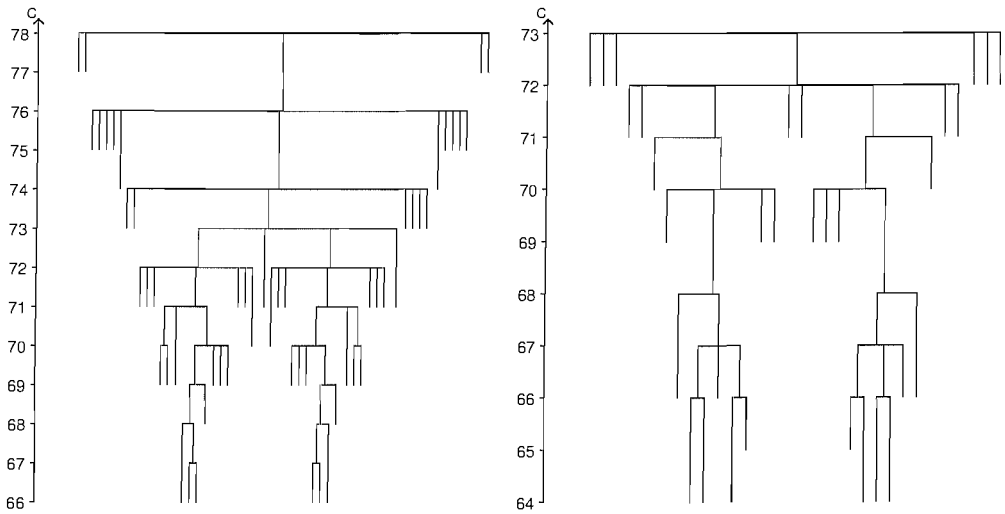
Figure 5.2 shows Barrier trees for instances of the Max-3-SAT problem for a fixed clause to variable ratio of $\alpha = M/N = 5$, slightly above the easy-hard transition at $M/N = 4.5$, and a varying number of variables. The 25 and 30 variable trees have only a partially evaluated landscape, as described in section 4.3. While we cannot guarantee the entire tree has been drawn, 10,000 descent runs have been run on each of the instances, and no local minima with a cost greater than our bound were found, giving us a good assurance that the entire tree has been evaluated; any local minima that do exist above the bound must have very small basins of attraction. Typically, the size of the tree, in both the number of local minima and the number of cost levels, increases with the number of variables.

5.1.3 Variations Over The Phase Transition

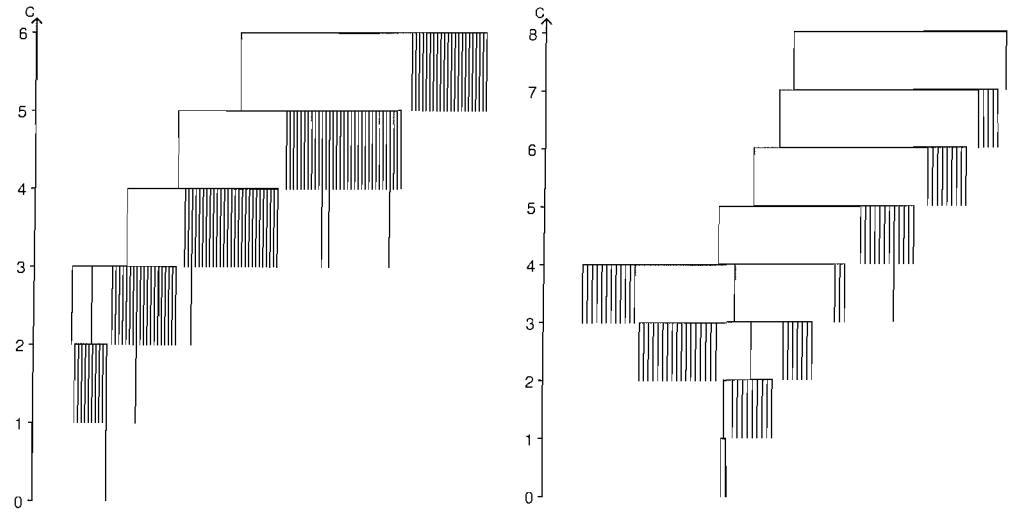
As discussed in section 2.3.2, randomly chosen instances of Max-3-SAT above the phase transition of $\alpha = M/N \approx 4.3$ seem to be fundamentally different from those below the phase transition. Barrier Trees may be a useful method for examining the effect this change has on the landscape.



(a) Max-3-SAT 20 variables, 90 clauses

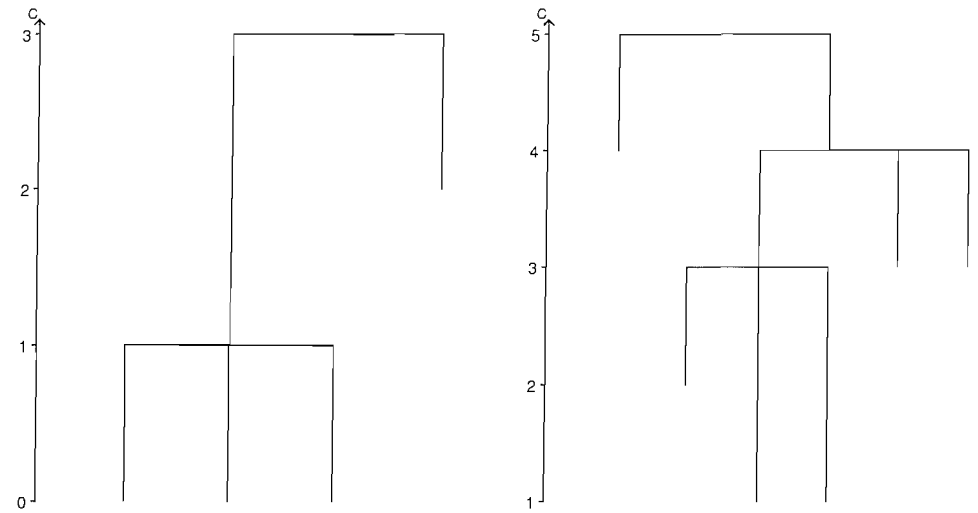


(b) Spinglass, 20 variables, fully connected.

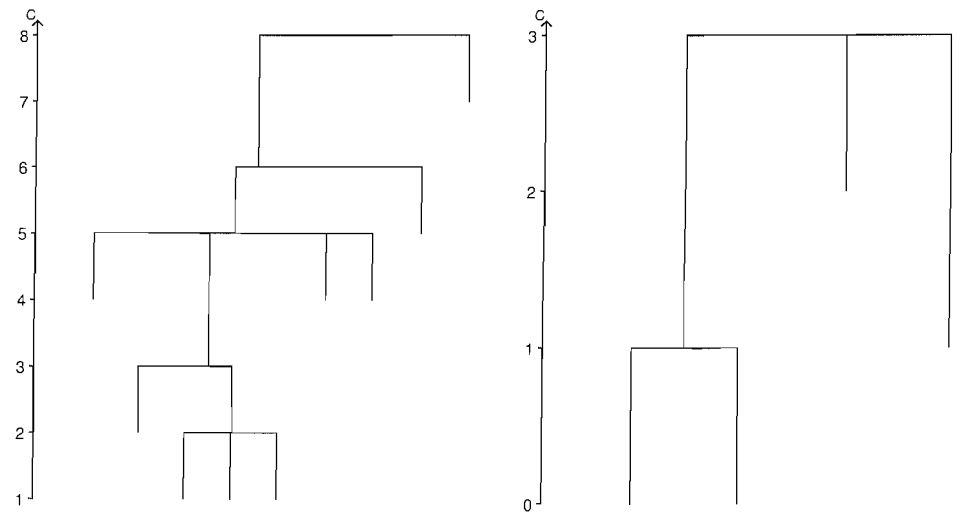


(c) Binary Perceptron, 13 variables 11 training examples.

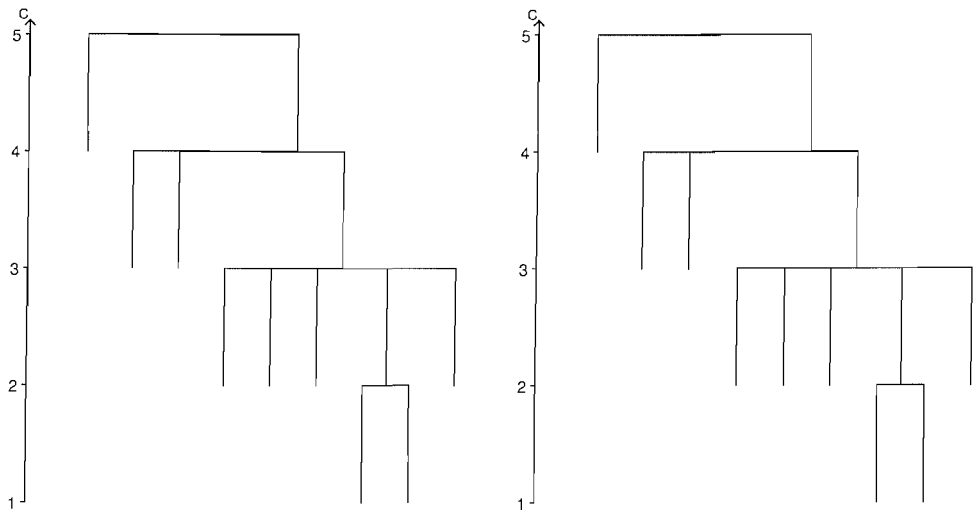
FIGURE 5.1: Examples of Barrier Trees from different problem classes.



(a) $N = 20$

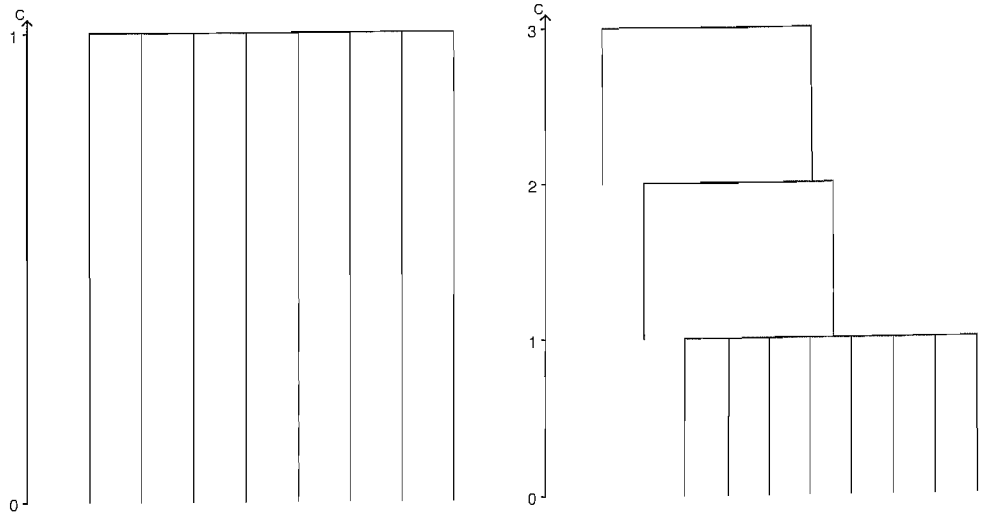


(b) $N = 25$

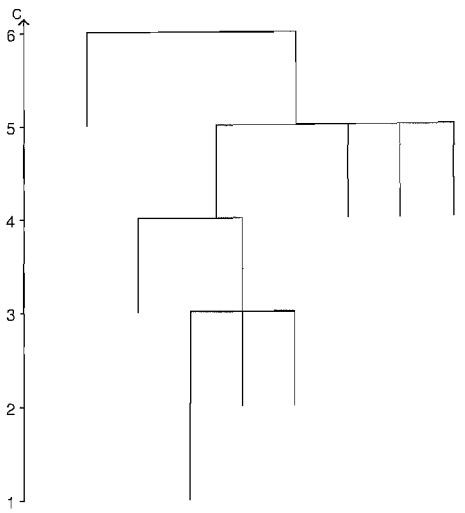


(c) $N = 30$

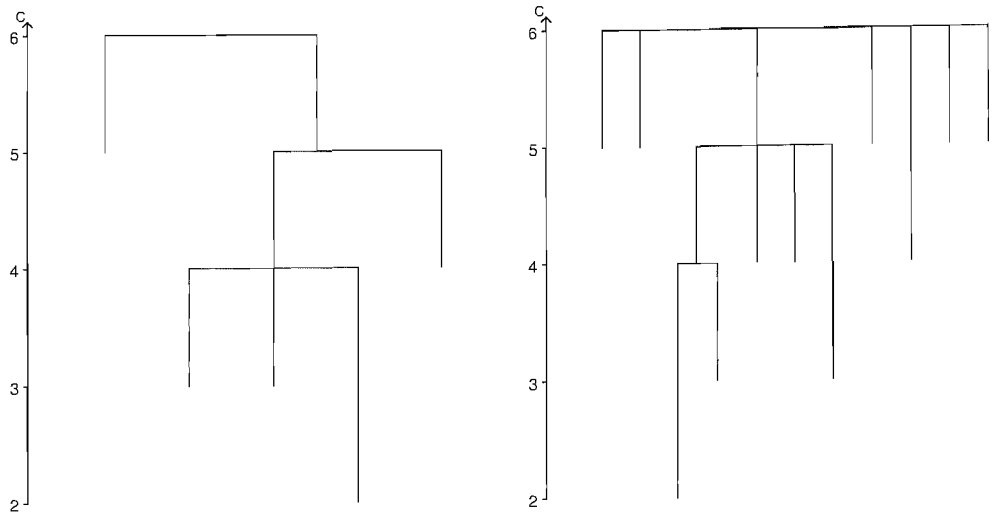
FIGURE 5.2: Barrier trees for instances of the Max-3-SAT problem with a varying number of variables, N. The clause to variable ratio is held constant at $\alpha = M/N = 5$



(a) $N/C = 2.5$ (b) $N/C = 3.5$



(c) $N/C = 4.5$



(d) $N/C = 5.5$ (e) $N/C = 6.5$

FIGURE 5.3: Barrier trees for instances of the Max-3-SAT problem for $N = 20$, with the ratio of clauses varying over the phase transition at $\alpha = M/N = 4.5$

Figure 5.3 shows Barrier Trees both sides of the phase boundary and at $\alpha = N/C = 4.5$. There seems to be an increase in complexity in the trees as the number of clauses increases, with more saddle points and local minima spread over a wider range of cost values. There is no evidence of a distinctive change at the phase transition. However, at the small size of problems we are looking at, the phase transition is less distinct, and can only be observed by averaging over a very large number of samples.

5.2 Statistics on a Single Instance

In addition to visualisation, Barrier Trees also provide a useful basis for gathering more detailed statistics for a more quantitative approach. There are measures that can be taken on the tree itself, such as the distribution of local minima and the average cost difference between saddle-points. And there are measurements based upon the partitioning, which allows a detailed examination of the basins and plateaus in the landscape.

We will now focus on a single instance to demonstrate some of these statistical measures. We have chosen a Max-3-SAT problem with $N = 30$ and $\alpha = M/N = 5$, shown in Figure 5.4. This landscape is too large to be completely evaluated, however it is relatively small compared to the largest landscapes that can be calculated, to increase the likelihood that all local minima have been captured. A large number of descent runs were run without finding any unevaluated local minima (see "Results of Descent", page 52). To further facilitate the presentation, an instance with only one global minimum was selected. Table 5.1 contains a number of statistical measurements, each of which is described in more detail below.

Sizes of Level-accessible Sets

A simple measure to take is the number of configurations in each level-accessible set, and this is shown in column 3 of table 5.1. Figure 5.5 shows that the number of configurations per cost level roughly follows an exponential growth as cost increases, except at a cost of 1, which has a lower than expected number of configurations. This appears to be an exceptional case; in the vast majority of problem instances seen, the number of configurations per cost level grows exponentially, as expected. Looking in more detail at the sizes of the local minima and saddle points in table 5.1, most of the exponential growth is in the saddle point at each cost level.

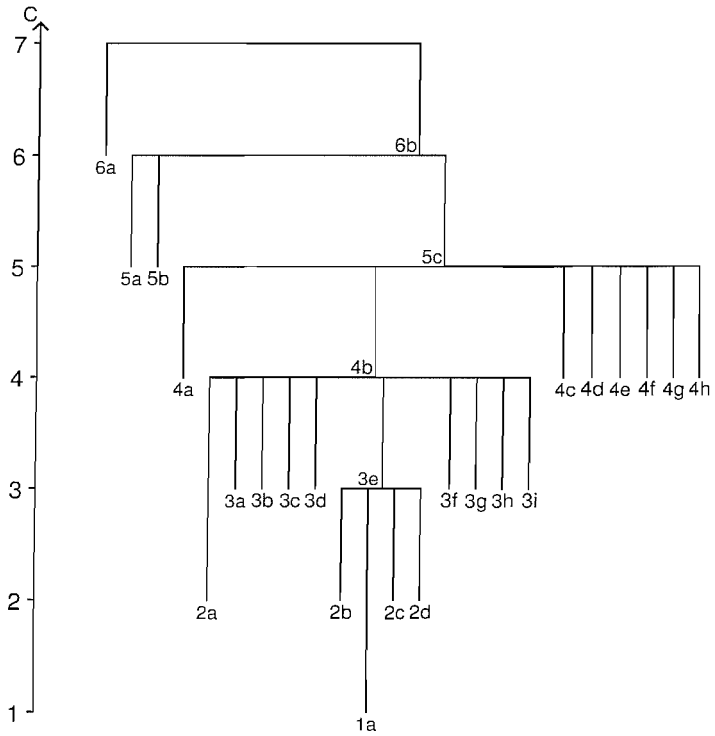


FIGURE 5.4: A Max-3-SAT problem, $N = 30$ and a clause to variable ratio of $\alpha = M/N = 4$. Statistics gathered on this problem are displayed in table 5.1

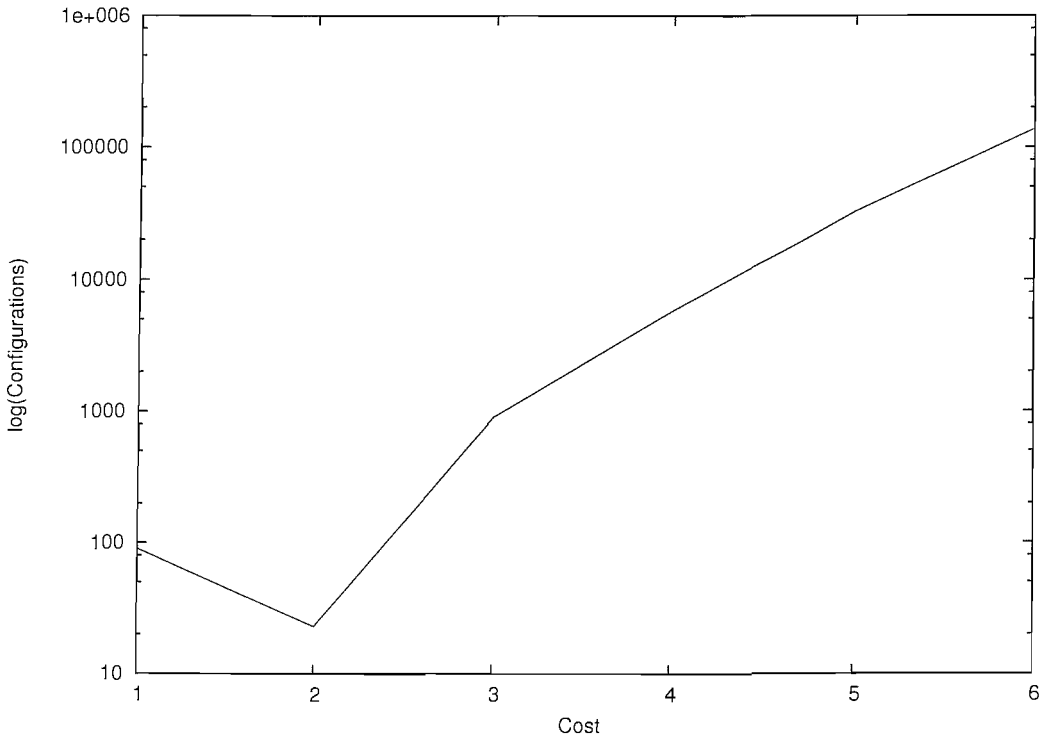


FIGURE 5.5: Number of configurations per cost level for the cost levels shown in table 5.1

Basin	Type	# of Configs.	Correlation with gm	Distance from gm	% Probability of Descent	Compaction Measure
1a	gm	90	0.728	0	27.9	0.93
2a	lm	9	−0.0577	10	4.94	0.99
2b	lm	8	0.458	4	7.36	0.98
2c	lm	2	0.347	5	7.95	0.93
2d	lm	4	0.516	2	2.41	1.00
3a	lm	4	0.249	6	1.51	1.00
3b	lm	4	0.218	6	3.85	1.00
3c	lm	6	0.0696	8	8.14	0.99
3d	lm	1	0.144	8	5.49	N/A
3e	s	871	0.590	1	N/A	0.78
3f	lm	1	−0.158	13	3.25	N/A
3g	lm	1	0.124	10	1.76	N/A
3h	lm	12	−0.0506	11	6.71	0.99
3i	lm	2	−0.000	11	1.68	1.00
4a	lm	1	−0.297	15	0.791	N/A
4b	s	5634	0.431	1	N/ A	0.52
4c	lm	5	−0.0114	10	1.57	0.99
4d	lm	74	0.171	5	8.58	0.89
4e	lm	1	0.103	10	0.290	N/A
4f	lm	3	−0.216	13	1.86	0.99
4g	lm	16	−0.0815	12	1.02	1.00
4h	lm	2	−0.167	14	0.956	1.00
5a	lm	1	−0.208	15	0.636	N/A
5b	lm	1	0.201	8	0.286	N/A
5c	s	32645	0.335	1	N/A	0.42
6a	lm	2	−0.212	15	0.121	1.00
6b	s	137519	0.281	1	N/A	0.39

TABLE 5.1: Details of the different basins of the problem represented in figure 5.4. Type is either gm for global minimum, lm for local minimum or s for saddle point.

Results of Descent

One measure of the basin of attraction of a local minimum is the probability of a descent algorithm reaching the minimum. This can be estimated by repeatedly running a descent algorithm, and counting how often it ends in each minimum. We use a simple greedy descent algorithm, modified so that it will only terminate in local minima (using our definition). This is done by making it explore plateau regions fully. This algorithm was repeated one million times. The percentage each minimum was reached is shown in column 6 of table 5.1. As there may be minima above the bound, it is possible this algorithm could terminate in a minimum we have not calculated. In the case of the problem represented in Figure 5.4, the descents always terminate

below the bound suggesting that all the local minima have been captured and the entire Barrier Tree has been evaluated.

Correlation in and Between Basins

An important aspect of the landscape is the closeness of configurations in different subtrees. There are a number of different measures we could use. One important measure is the average correlation. The correlation between two binary strings $\boldsymbol{x}$ and $\boldsymbol{x}'$ is equal to

$$q(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{N} \sum_{i=1}^{N} (2x_i - 1)\,(2x'_i - 1).$$

where we use the convention that $x_i = 1$ if it is true and $x_i = 0$ otherwise. If the strings are identical $q = 1$, while if they are anti-correlated (i.e. $x'_i = \neg x_i$ for all i) then $q = -1$. The correlation function is related to the Hamming distance, H, by $H = N(1 - q)/2$. The mean correlation between two sets of configurations, $\mathcal{P}_i$ and $\mathcal{P}_j$ is given by

$$q(\mathcal{P}_i, \mathcal{P}_j) = \frac{1}{|\mathcal{P}_i|\,|\mathcal{P}_j|} \sum_{\boldsymbol{x} \in \mathcal{P}_i} \sum_{\boldsymbol{x}' \in \mathcal{P}_j} q(\boldsymbol{x}, \boldsymbol{x}').$$

This provides a measure of the average distance between the two sets of configurations. The fourth column of table 5.1 shows the correlation between the basin of the global minimum and the other basins in the Barrier tree.

Compaction

An idea that can be used to help understand the shape of a basin is that of compaction. That is, whether a basin is a stringy structure, stretched out over many sides of the hypercube, or tightly compressed into a small number of sides. We have used a measure of compaction based upon the single set correlation

$$q(\mathcal{P}_i) = \frac{1}{|\mathcal{P}_i|\,(|\mathcal{P}_i| - 1)} \sum_{\boldsymbol{x} \in \mathcal{P}_i} \sum_{\substack{\boldsymbol{x}' \in \mathcal{P}_i \\ \boldsymbol{x}' \neq \boldsymbol{x}}} q(\boldsymbol{x}, \boldsymbol{x}').$$

Note that we do not include strings correlating with themselves.

While this may appear to give a good measure of compactness, it does not fully take into account the size of the set. Larger sets end up with a lower self correlation because they contain more strings, and those strings must therefore differ by a greater degree. To compensate for this, we calculate the maximum possible correlation of a connected set of the same size. The maximum self correlation occurs when as few

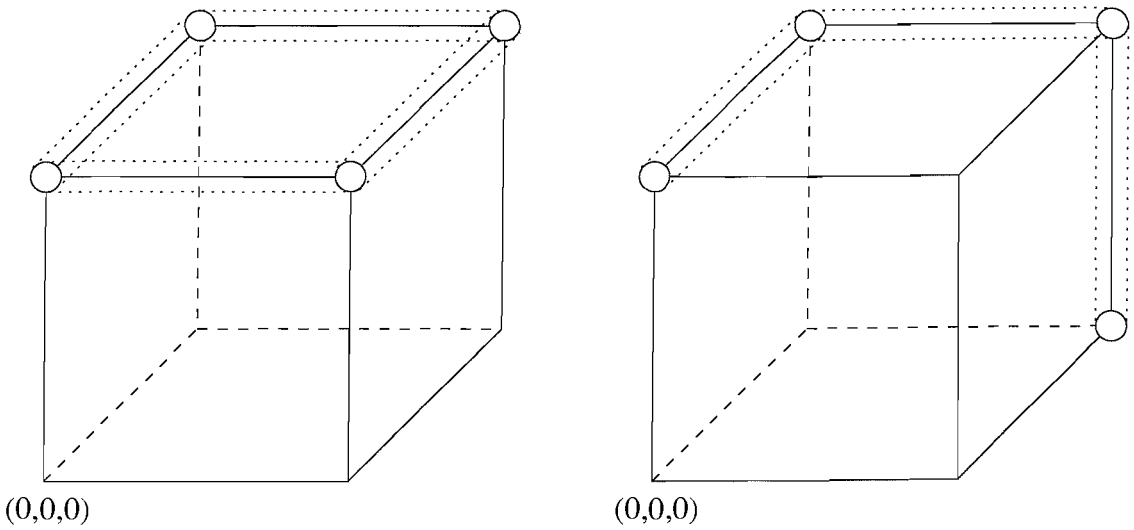


FIGURE 5.6: Example of a compact (l.h.s.) and a non-compact (r.h.s) set of configurations on a 3D cube, the topology of a 3 binary variable landscape.

sides of the hypercube are occupied as possible, so that there is a minimal variation in the binary variables describing the different configurations. The minimum number of binary variables that need to vary to describe a set of size $|\mathcal{P}_i|$ unique configurations is given by $\log_2 |\mathcal{P}_i|$. The maximum self correlation of a set of size $|\mathcal{P}_i|$ is therefore given by

$$q_{max}(|\mathcal{P}_i|) = \frac{|\mathcal{P}_i|}{N(|\mathcal{P}_i| - 1)}(N - \log_2 |\mathcal{P}_i| - 1).$$

Our measure of compactness then becomes the fraction $q(\mathcal{P}_i)/q_{max}(|\mathcal{P}_i|)$. A value of 1 means that the basin is as compact, and occupying as few sides of the hypercube, as possible.

Compaction is an interesting measure because it relates directly to uniform crossover. Each variable of the result of a uniform crossover between two configurations can only have a value that at least one of it's parents have for the same variable. If both parents have an equal value for a particular variable, the child will have the same value. Therefore highly correlated parents will produce a child that is highly correlated to both parents. A high compaction value over a set of configurations therefore indicates that uniform crossover between randomly selected members of that set is likely to be a member of the same set.

The compaction of the basins is shown in the last column of table 5.1. Note that there is no compaction measure for a basin of size one, and basins of size two always have a compaction of 1. Most minima are compact. This is perhaps a surprising empirical observation. The saddle-points are non-compact, but these can be composed of many disconnected level-connected sets.

Basin	Percentage result of crossover
Off tree	7.436
6b	12.785
5c	27.350
4d	52.428

TABLE 5.2: The percentage of the results of uniform crossover of randomly chosen configurations from basin 4d, from the problem in figure 5.4. Off tree are the crossovers that resulted in a configuration not mapped to the tree

Minimum Hamming Distance

Another measure giving an idea of closeness between different subtrees of the Barrier tree is the minimum Hamming distance. That is, the smallest number of bit-flips required to get from a configuration in one level-accessible set to a configuration in a different level-accessible set. We show the minimum Hamming distance between each level-accessible set and the global minimum in column five of table 5.1. The local minima are typically quite distant from the global minimum. However, the saddle-points usually contain at least one configuration very close to the global minimum. This is not surprising when the size of the saddle-points is taken into account.

Crossover

An operator vitally important to Genetic Algorithms is crossover. A simple way to gain understanding of Genetic Algorithms is to look at how crossover performs within a basin. A more complex model of crossover is explored in chapter 6.

We performed 10 000 000 uniform crossovers (defined in section 3.1.1) on members selected randomly from basin 4d, from the problem in Figure 5.4. This basin was chosen as it is one of the larger local minima which is not the global minimum. The percentage of times the result of each crossover landed in a particular basin is shown in table 5.2.

As can be seen, most crossovers resulted in a configuration within the same basin, which agrees with the high compaction value of basin 4d (see table 5.1). No child produced by crossover ever jumped across or down the Barrier tree. This, supports the hypothesis that the Hamming neighbourhood is an appropriate neighbourhood to use when using Barrier Trees to study uniform crossover.

5.3 Animated Search Heuristics

Our definition of Barrier Trees and the partitioning of the landscape implicitly defines a mapping from every configuration to a vertex on the tree. One use of this mapping is to animate a search algorithm. In this section we show the animation of a Genetic Algorithm and a simulated annealing algorithm

5.3.1 Genetic Algorithm

Figure 5.7 shows several stages from a run of a genetic algorithm. Each level accessible-set containing a member of the genetic algorithms population is marked with a dot. Often several members are in the same level-accessible set, so are represented by only a single dot, but the software allows the number of candidate solutions represented by a single dot to be determined interactively.

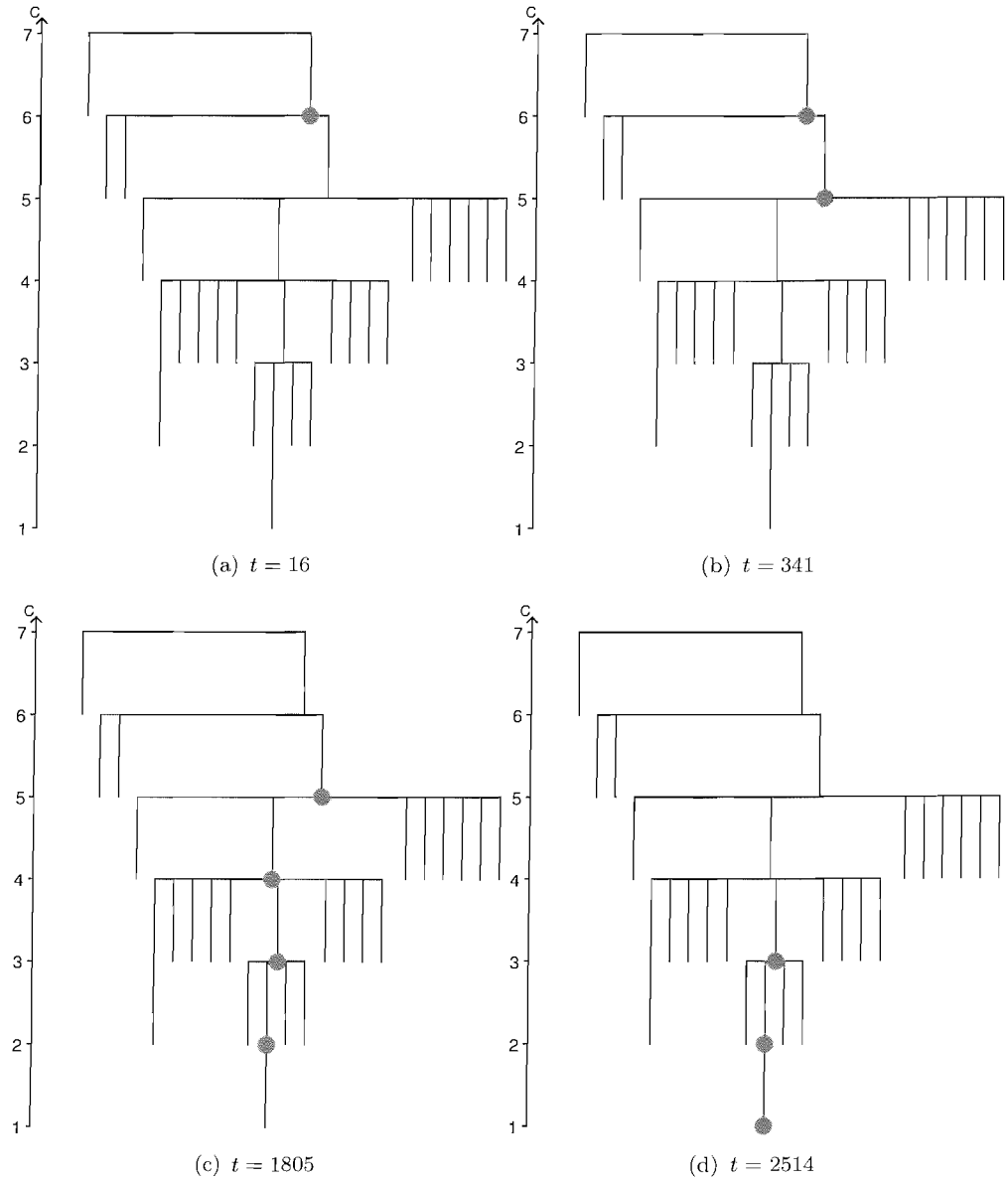
The GA run is of a simple steady-state GA with a population size of 10. Binary tournament selection is used with crossover and a mutation probability of 0.1. The problem is a 20 variable Max-3-SAT problem with 120 clauses. In this case it can be seen that the algorithm descends straight down the tree, without getting trapped in local minima. A more detailed view of the algorithm can be obtained interactively in the program, however in print we are limited to just a few 'snapshots' of an algorithms run.

5.3.2 Simulated Annealing

A simple simulated annealing algorithm with a fixed temperature of $T = \frac{1}{3}$ was run on a 40 variable Max-3-SAT problem with 180 clauses. Figure 5.8 shows the location of the search at various intervals throughout the run of the algorithm. It can be seen that the algorithm moves to lower cost solutions. What is less clear from these snapshots is the amount of time spent moving back and forth to higher cost solutions.

5.4 Conclusions

Barrier Trees are a useful visualisation and analysis tool for looking at small instances of combinatorial optimisation problems. Distinct visual characteristics can be identified, and the data provided by Barrier Trees are a useful basis to calculate many different statistics.

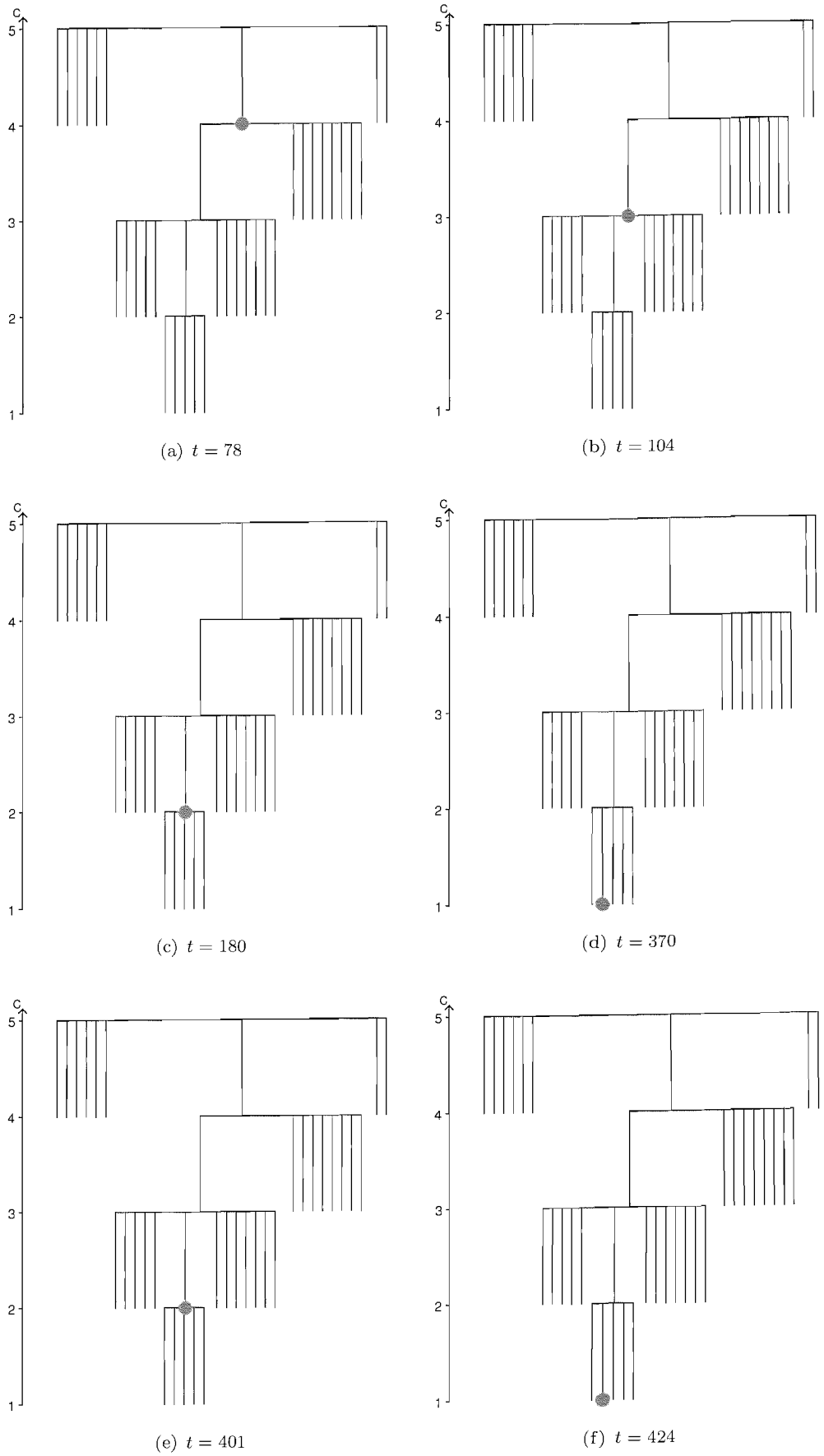


(a) $t = 16$ (b) $t = 341$

(c) $t = 1805$ (d) $t = 2514$

FIGURE 5.7: A Genetic algorithm working on Max-3-SAT problem. t is the iteration of the algorithm.

There are many different ways the visualisation could be improved. It would be useful to have some representation of the size of each level-accessible set on the tree. An obvious way to represent this would be circles or bars at each node, with a size proportional to their corresponding node. It is likely a log scale would provide a more useful scaling for this. It would also be useful to have some idea of the number and distribution of sizes of the level-connected sets making up each level-accessible set; in Max-SAT, most level-accessible sets are made up of one large level-connected set and several very small (1 or 2 configuration) ones, but this information cannot be seen on the Barrier Tree, and is relevant to understanding the landscape represented



(a) $t = 78$ (b) $t = 104$

(c) $t = 180$ (d) $t = 370$

(e) $t = 401$ (f) $t = 424$

FIGURE 5.8: A simulated annealing algorithm working on a 40 Max-3-SAT problem with a fixed temperature of $T = 3$.

by the tree.

Looking at underused resources in our Barrier Tree figures, there are two main visual resources which remain unused: colour and the x-axis, which currently has no meaning. Obvious uses for these would be the colouring of nodes to indicate size, and ordering the x-axis to give some idea of the proximity of different level-accessible sets to each other. These ideas obviously need exploring, and it is likely that different approaches would be more useful for different situations.

The animation of search heuristics is quite basic. In addition to relatively simple additions such as showing how many members of a population are in a single level-accessible set, it would be useful to have some method to display many different repetitions of an algorithm effectively. This would make it far easier to gain an understanding of an algorithm's behaviour. Currently, it is necessary to compare many different animations of different runs of an algorithm to try and pick out common features; a difficult and time consuming process.

Barrier Trees are useful as a basis for analysis. The partitioning makes it easy to collect many statistics, such as the size of basins, the number of local minima and the sizes of the barriers between them. In combination with the visual representation of the landscape, Barrier Trees are a flexible and powerful tool.

Chapter 6

Barrier Based Problem Models

Model problems are one of the key tools used to study the complexity of heuristic search on combinatorial optimisation problems. These models are designed to test theories of what features and properties are important to algorithm design, but are constructed in a way that makes them amenable to numerical analysis. This is the approach behind several 'toy' problems, such as Ones Counting, the Hurdle problem (Prügel-Bennett, 2004b), the Royal Road problem (Mitchell et al., 1992b), De Jong's test-bed function (De Jong, 1995) and the H-IFF problem (Watson, 2001). These models have an analytical solution, and have been effectively used to show particular features and behaviours of different algorithms. The problem with this approach is that the relationship to practical problems faced by practitioners is unknown. The models tell us nothing about the types of problems on which heuristic search algorithms are actually used on. Even if the properties the model problem is designed to show exist in practical problems, it cannot be said whether they are dominant properties influencing algorithm performance on those practical problems.

In this chapter we present a technique for generating model problems from small instances of $\mathcal{NP}$-hard problems that are considered examples of the problems typical of practical settings. The goal is not to create an accurate predictive model but to create numerically analysable models that have a well defined relationship to challenging practical problems, so we can learn about these problems. From these models we hope that conclusions can be drawn on algorithm behaviour on large, difficult problems.

We call the models described in this chapter *Barrier Based* models, in a reference to Barrier Trees. They consist of a partitioning of the landscape and an approximation of the Neighbourhood function acting on a configuration within a partition. For this the principal partitioning we use is that of level-accessibility, defined in chapter 4.

This partitioning is useful as it results in a greater reduction in states than other partitionings while maintaining the separations between basins in the landscapes; this is why the models are described as Barrier Based. Other partitionings of the landscape can be used, and we explore level-connected sets as an alternative.

Different heuristic search algorithms can be modelled on the Barrier Based models. The model provides a neighbourhood function, so all algorithms that use a simple neighbourhood, such as descent and simulated annealing can be "run" on the model problems. The extent to which such simple algorithms can be numerically analysed depends on the complexity and form of their navigation strategies. While search operators with more complex neighbourhoods are more difficult to represent in this framework, they can be approximated by careful use of a simpler neighbourhood. In chapter 8 we explore a simple way of modelling crossover using a Barrier based model with a Hamming neighbourhood.

Combinatorial search algorithms lend themselves to analysis as simple Markov chains. An analytical solution can often be written for a search algorithm working on a problem directly, although the number of states in the Markov chain is normally proportional to the size of the search space, and therefore massively too large for any useful calculation. A Markov chain analysis has only been performed on very small problems (e.g Hoffmann and Salamon (1990)). By partitioning the search space, our model massively reduces the number of states needed in the Markov chain, making the Markov chain analysis practical. A Markov chain analysis is the technique we use to analyse algorithm behaviour on the Barrier Based models; this is the tool the models were designed around.

In this chapter, we give an overview of previous work that has used Markov models to model heuristic search algorithms. This is followed a definition of our Barrier Based models and a demonstration of how simple descent can be modelled within this framework. We then go through several calculations on Markov chains that we will use. Chapter 7 presents results comparing descents on the original problem instance and on the model, and then contains an analysis of the differences in algorithm. Finally, chapter 8 discusses the limitations of the types of problem that can be modelled, and explores a couple of techniques that reduce these limitations.

6.1 Markov Chains and Heuristic Search

The use of Markov chains as a tool for modelling heuristic search has been of particular interest in the genetic Algorithm community. An important early work was Goldberg and Segrest (1987), which was a dynamic model of a genetic algorithm for 2 bit

strings. This was an infinite population model, sometimes called an exact model. By taking the population to be infinite in size, the dynamics of the model become stable, allowing an exact calculation of the expected behaviour to be made. Goldberg's model was further developed in Bridges and Goldberg (1987); Whitley (1993). Vose and Liepins (1991) presented a new independently created infinite population model, which also included mutation. This model quickly became very popular, and perhaps the best known work on infinite population models for Genetic Algorithms is Vose (1999).

While infinite population models are very powerful, they are limited because finite population effects are very important, both in explaining Genetic algorithm behaviour and for design of Genetic algorithms. As a trivial example, an infinite population model provides no clue as to what size population a Genetic algorithm should have to solve a problem most efficiently. There has consequently been interest in finite population models. One of the first papers modelling a finite population GA with a Markov chain was Nix and Vose (1992). This used fixed-length binary strings, 1-point crossover, bit-flipping mutations and fitness proportional selection. This model requires a state for every possible population, and the number of possible populations becomes large very quickly with increasing population size and the number of variables. For a population of size P on a problem with N binary variables, the number of possible populations, as calculated in Nix and Vose (1992) is

$$\frac{(2^N + P - 1)!}{P!(2^N - 1)!}.$$

Clearly, it is almost impossible to work with all but the smallest problems and populations using this model directly.

To tackle this problem, many researchers have looked at clustering or coarse graining techniques. Spears and de Jong (1996) was perhaps the first paper to look at this technique in the Genetic Algorithm community, with a novel algorithm described in more detail in Spears (1998). This technique merges Markov states directly, and can be applied to any Markov chain representation. Many other researchers have looked at coarse graining for example Rattray (1996), Rowe (1998) and van Nimwegen et al. (1999). Some of these techniques rely on knowledge of the model gained from knowing the algorithm and problem being modelled, for example Rowe and Moey (2004). It is into this latter category of coarse graining techniques that the model presented here falls into.

6.2 Definitions

The basis of our models is to partition the landscape, and to make the approximation that a search heuristic will treat each configuration within a partition identically. We then calculate an approximation of the neighbourhood function acting on each partition, which is an average of the Neighbourhood function working on all the configurations within that partition. The result is a model of the problem that maintains the connectivity relation between the different partitions of the landscape, but has a massively reduced state space and is therefore more amenable to analysis as a Markov chain.

We begin by defining our model of the neighbourhood function; this remains the same whatever partitioning is used. We discuss how the landscape can be partitioned in section 6.2.2. For our purposes it is sufficient that some partitioning exists, in addition to a cost landscape, defined in chapter 2 as consisting of a set of configurations $\mathcal{C}$, a cost function $f : \mathcal{C} \rightarrow \mathbb{R}$ and a neighbourhood function $N : \mathcal{C} \rightarrow 2^{\mathcal{C}}$. We take each partition to be a set of configurations, and label them $\mathcal{P}_1$ to $\mathcal{P}_n$. A complete partitioning is assumed, so that every configuration is a member of one, and only one, partition.

We wish to model the connective relationships between the different partitions under the neighbourhood relation. We do this as a probability distribution for each partition. For each partition, the distribution gives the probability that a randomly chosen neighbour will be in each partition. To do this we make two approximations,

- Search algorithms choose neighbours randomly; they have no information about a neighbour before picking it and evaluating its cost.
- The neighbours of every configuration are as likely to be chosen as the neighbours of any other configuration within that partition; every configuration is as likely to be visited by a search algorithm as any other configuration within the partition.

By making these approximations we can calculate the connectivity between different partitions as being the proportion of neighbours of the configurations in one partition that are in another partition. This proportion gives us a probability that a neighbour picked at random is in the other partition.

We represent these probabilities in a matrix T, where T_{ij} gives the probability that picking a neighbour of a configuration in partition $\mathcal{P}_i$ is in partition $\mathcal{P}_j$; each row of the matrix is a separate probability distribution. These distributions are calculated

as a summation of the probabilities of a neighbour being in each partition across all the configurations within the partition. More precisely if we define a matrix of neighbour counts, **N**

$$N_{ij} = \sum_{c \in \mathcal{P}_i} |N(c) \cap \mathcal{P}_j|,$$

then matrix we use in our model can be expressed as

$$T_{ij} = \frac{N_{ij}}{\sum_{l=1}^{n} N_{il}}.$$

A matrix like this can be seen as a transition matrix, the basis of most Markov chain analysis. We will describe some calculations that can be performed on Markov chains in section 6.3.

It is also useful to have some fixed probability distribution. Almost all search algorithms start by choosing configurations at random. Assuming this is the case we can calculate the probability distribution at this time $t = 1$ exactly as

$$p_i(1) = \frac{|\mathcal{P}_i|}{|\mathcal{C}|}.$$

6.2.1 Modelling Simple Descent

Simple descent is the most basic type of heuristic search algorithm. Here we demonstrate how it can be represented on a Barrier model in a form which is easy to analyse.

If we take the probabilities of our model, calculated above, as transition probabilities (i.e. a probability that we have a change in state from one partition to another), then the model can be seen as modelling a random walk. Neighbours are chosen at random and accepted. Simple descent is very similar, with the only difference being that moves to higher cost states are rejected. Therefore, to model descent, all that is needed is to reduce transition probabilities from lower cost states to higher cost states to 0, and increase the the probability of staying in the same state (T_{ii}) by the same amount. Formally, we define this descent transition matrix, **D** as

$$D_{ij} = \begin{cases} T_{ij}[f(\mathcal{P}_j) \leq f(\mathcal{P}_i)]_0^1 & \text{if } i \neq j \\ T_{ij} + \sum_{l=1}^{n} T_{il}[f(\mathcal{P}_l) > f(\mathcal{P}_i)]_0^1 & \text{if } i = j \end{cases}$$

where $[pred.]$ is notation for

$$[pred.]_0^1 = \begin{cases} 1 & \text{if } pred. \text{ is true} \\ 0 & \text{if } pred. \text{ is false.} \end{cases}$$

If we take the starting distribution, $\boldsymbol{p}(1)$, to be simply proportional to the size of each partition, as described above, our model predicts the distribution at time t as being

$$\boldsymbol{p}(t)^T = \boldsymbol{p}(t-1)^T \mathbf{D} = \boldsymbol{p}(1)^T \mathbf{D}^{t-1}.$$

This can be seen by expanding the matrix multiplication; this is the basic Markov chain calculation.

6.2.2 Partitioning

The partitioning used obviously has a large effect on the behaviour and accuracy of the model. It is possible to choose almost any partitioning, and it is by no means clear what properties are desired. We have explored the use of two different partitionings, both of which come from our work on Barrier Trees: level-accessible sets and level-connected sets (chapter 4).

Level-connected sets are perhaps the most obvious partitioning to use. Connected configurations of equal cost are partitioned and approximated at being equal. The number and cost of local minima are maintained, and the connectivity between different minima and parts of the landscape are not disrupted by the partitioning.

While level-connected sets may have a very close relationship to the original landscape, the number of level-connected sets in a landscape can be very large, leading to a model that is impracticable to analyse as a Markov chain. For this reason, we have also used level-accessible sets as a partitioning scheme. These share many of the useful properties of level-connected sets (e.g. the local minima remain unchanged) but often result in dramatically fewer partitions. For example, in a typical 20 variable Max-SAT problem, around 100 level-accessible sets partition the landscape, compared to around 300 level-connected states. An extreme example is Ones-counting, where each level-connected set contains only one configuration, leading to 2^N states, but is partitioned completely by only N level-accessible sets. However, level-accessible sets are not necessarily connected, and it is therefore possible that modelling them as a single state within our model will allow search heuristics to make 'jumps' between different parts of the landscape that are not possible on the original problem. An empirical comparison of the accuracy of the two different models is made in chapter 7.

With both these partitionings, every partition contains configurations with only one cost. This is useful for the problem classes we have examined, which have many configurations with an equal cost. Since most search heuristics do not distinguish between configurations of an equal cost, the behaviour within a single partition should be a random walk. If different cost configurations exist within a partition, for most algorithms the behaviour becomes far more complex within that partition. It also becomes more difficult to model the transitions between partitions without a definite cost; consider the model of descent described above, where transition probabilities are derived from a comparison of cost. If each partition does not have a uniform cost, it becomes far more difficult to make that comparison. However, for problem classes with a huge range of costs such as the Travelling Salesman Problem, having only a single cost in each partition will result in far too many states. How to partition these classes of problems is an open problem.

6.3 Markov Chains

A Markov chain (Kemeny and Snell, 1960) is a stochastic process with discrete time-steps where the probability of being in any state is influenced only by the previous state. They are consequently a useful tool for studying combinatorial search algorithms (e.g. Hoffmann and Salamon (1990)). This section describes the basic definitions and concepts of Markov chains. It then proceeds to the derivations of the calculations we plan to use. These calculations are a small part of a well established framework, and the derivations are included here only for completeness.

6.3.1 Definitions

We take a Markov chain as consisting of a set of states, and transition probabilities from each state to any other. The transition probability between states i and j is the probability that the system will be in state j at time $t+1$ if it was in state i at time t. The transition probabilities are conveniently arranged in a matrix, $\mathbf{M}$, such that M_{ij} is the probability of a transition from i to j in a single time step. A row from this matrix sums to 1.

When performing Markov chain calculations, we are normally interested in the probability distribution over the states at some time t. We represent this as a vector $\boldsymbol{p}(t)$. Note that the matrix of transition probabilities has been defined so that

$$\boldsymbol{p}^T(t+1) = \boldsymbol{p}^T(t)\mathbf{M}.$$

It is often useful to have a fixed probability distribution, which we take to be at $t = 1$ instead of the more usual $t = 0$. Defining our start point at $t = 1$ means that for most algorithms t is also a count of function evaluations.

Markov chains are described as having certain properties. A Markov chain is *irreducible* if every state can be reached from every other state; that is, there is a chain of non-zero probability steps from every state to every other state. A process is *periodic* if there is some state to which the process will continually return with some fixed time period greater than 1. The return time of a state is the time taken to return to the state after leaving it. A Markov chain is *ergodic* if it is irreducible and the expected return time for every state is finite. An ergodic Markov chain has a single *stationary distribution*, a probability distribution that remains unchanged after any number of time steps.

Stationary Probability Distribution

If a Markov chain is ergodic, then it is known that it has one stationary probability distribution, that does not change after further time steps. In this section, we show how the stationary distribution can be derived numerically, and derive a measure of how fast probability distributions tend to the stationary distribution as t tends to infinity.

The probability distribution at any time t can be expressed as

$$\boldsymbol{p}^T(t) = \boldsymbol{p}^T(t-1)\mathsf{M} = \boldsymbol{p}^T(1)\mathsf{M}^t.$$

We define the eigenvectors of M so that the left and right handed eigenvectors are $\boldsymbol{v}_i$ and $\boldsymbol{u}_i$ respectively, such that

$$\begin{aligned} \boldsymbol{v}_i^T\mathsf{M} &= \lambda_i \boldsymbol{v}_i^T, \\ \mathsf{M}\boldsymbol{u}_i &= \lambda_i \boldsymbol{u}_i, \\ \boldsymbol{v}_i^T \boldsymbol{u}_j &= [i = j]_0^1. \end{aligned}$$

Every left handed eigenvector is linearly independent, and provided M is not defective there are at least n of them, so any vector $\boldsymbol{p}$ can be written as a sum of multiples of $\boldsymbol{v}$. We therefore write $\boldsymbol{p}(1)$ as

$$\boldsymbol{p}(1) = \sum_{l=1}^{n} c_l \boldsymbol{v}_l$$

It follows that our expression for $\boldsymbol{p}(t)$ for any time t can be written as

$$\boldsymbol{p}^T(t) = \sum_{l=1}^{n} c_l \boldsymbol{v}_l^T \mathbf{M}^t.$$

From the definition of eigenvectors this can be written as

$$\boldsymbol{p}^T(t) = \sum_{l=1}^{n} c_l (\lambda_l)^t \boldsymbol{v}_l^T$$

If the Markov chain is ergodic, there will be only one pair of eigenvectors with a corresponding eigenvalue of 1, and all others will be smaller. Assuming this is the case, we label these eigenvectors so that $\lambda_1 = 1$, and label the other eigenvector pairs in descending order of eigenvalue magnitude. The above expression can be rewritten as

$$\boldsymbol{p}^T(t) = c_l \boldsymbol{v}_l^T + \sum_{l=2}^{n} c_l (\lambda_l)^t \boldsymbol{v}_l^T.$$

The stationary distribution can be found by considering what happens as t tends to infinity. If ergodic, for all $l > 1$, $|\lambda_l| < 1$, therefore as t tends to infinity, the summation tends to 0. Therefore,

$$\boldsymbol{p}^T(\infty) = c_1 \boldsymbol{v}_l^T.$$

The stationary distribution is proportional to the left-handed eigenvector corresponding to an eigenvalue of 1.

From the above equations, it can be seen that the second largest eigenvalue, λ_2, is an indicator of how quickly any probability distribution tends to the stationary distribution. If this is small then $(\lambda_l)^t$ tends to 0 quickly for all $l > 1$. We use this to measure the rate of convergence, or mixing rate. At time t, the λ_2 component of the probability distribution can be expressed as

$$\lambda_2^t = \boldsymbol{e}^{(t \log(\lambda_2))}.$$

As $\lambda_2 < 1$, this is an exponential decay. We take the mixing time, T_m, to be the time it takes for this component to reduce by multiple of $\boldsymbol{e}^{-1}$. Therefore

$$T_m = \frac{-1}{\log |\lambda_2|}.$$

Note that we take the magnitude of λ_2, as it may be a complex number.

Mean First-Passage Time

The mean first-passage time is the average time until a state is visited for the first time. The calculations shown here to calculate the mean first-passage time efficiently from some starting distribution are taken directly from Prügel-Bennett (2004b), with only minor adjustments.

We take the set $\mathcal{G}$ to be the set of target states for which we are calculating the mean-first passage time. We consider the modified transition matrix $\hat{\mathbf{M}}$,

$$\hat{\mathbf{M}}_{ij} = \begin{cases} \mathbf{M}_{ij} & \text{if } i \notin \mathcal{G} \\ 0 & \text{otherwise} \end{cases}$$

The target states have become an absorbing boundary. The eigenvalues of this matrix are all strictly less than 1 as probability is not maintained. For convenience we define a vector $\boldsymbol{\delta}$,

$$\delta_i = \begin{cases} 1 & \text{if } i \in \mathcal{G} \\ 0 & \text{otherwise} \end{cases}$$

If $\hat{\boldsymbol{p}}^T(t) = \boldsymbol{p}^T(1)\hat{\mathbf{M}}^{t-1}$, then $\boldsymbol{\delta}^T\hat{\boldsymbol{p}}(t)$ is the probability a target state is reached for the first time at time t, and $\mathbf{1}^T\hat{\boldsymbol{p}}(t)$ (i.e. the sum of the elements in $\hat{\boldsymbol{p}}(t)$) is the probability that none of the goal states have been reached *before* time t. The mean first-passage time is equal to

$$T_{fpt} = \sum_{t\geq 1} t\hat{\boldsymbol{p}}_1(t) = \sum_{t\geq 1} t\boldsymbol{p}^T(1)\hat{\mathbf{M}}^{t-1}\boldsymbol{\delta}.$$

To simplify the calculation, we first derive an identity. We use the closed form expression for the geometric series

$$\sum_{t=1}^{N} \hat{\mathbf{M}}^{t-1} = (\mathbf{I} - \hat{\mathbf{M}})^{-1}(\mathbf{I} - \hat{\mathbf{M}}^N).$$

This holds for any matrix provided $(\mathbf{I} - \hat{\mathbf{M}})$ is non-singular, which can be shown by pre-multiplying both sides by $(\mathbf{I} - \hat{\mathbf{M}})$. Since the magnitudes of the eigenvalues $\hat{\mathbf{M}}$ are all less than 1,

$$\lim_{N\to\infty} \hat{\mathbf{M}}^N = 0$$

thus

$$\sum_{t\geq 0} \hat{\mathbf{M}}^{t-1} = (\mathbf{I} - \hat{\mathbf{M}})^{-1}.$$

Since eventually an absorbing state will be reached

$$\left(\sum_{t\geq 1} \boldsymbol{p}(1)^T \hat{\mathbf{M}}^t\right) \boldsymbol{\delta} = 1 = \boldsymbol{p}(1)^T (\mathbf{I} - \hat{\mathbf{M}})^{-1} \boldsymbol{\delta}.$$

Since this holds for any $\boldsymbol{p}(1)$ for which $\boldsymbol{p}^T(1)\mathbf{1} = 1$ we can deduce the identity $(\mathbf{I} - \hat{\mathbf{M}})^{-1}\boldsymbol{\delta} = 1$.

Returning to the mean first-passage time

$$\begin{aligned} T_{fpt} &= \left(\sum_{t\geq 1} t\boldsymbol{p}^T(1)\hat{\mathbf{M}}^{t-1}\right)\boldsymbol{\delta} \\ &= \boldsymbol{p}^T(1)\left(\frac{\partial}{\partial \hat{\mathbf{M}}}\sum_{t\geq 1}\hat{\mathbf{M}}^t\right)\boldsymbol{\delta} \\ &= \boldsymbol{p}^T(1)\left(\frac{\partial}{\partial \hat{\mathbf{M}}}((\mathbf{I} - \hat{\mathbf{M}})^{-1} - 1)\right)\boldsymbol{\delta} \\ &= \boldsymbol{p}^T(1)(\mathbf{I} - \hat{\mathbf{M}})^{-1}(\mathbf{I} - \hat{\mathbf{M}})^{-1}\boldsymbol{\delta} \\ &= \boldsymbol{p}^T(1)(\mathbf{I} - \hat{\mathbf{M}})^{-1}1 \end{aligned}$$

where we used the identity $(\mathbf{I} - \hat{\mathbf{M}})^{-1}\boldsymbol{\delta} = 1$. Thus the mean first-passage time can be calculated by inverting the matrix $\mathbf{I} - \hat{\mathbf{M}}$.

Chapter 7

Accuracy of Barrier Based Models

In this chapter Barrier Based models of problem instances are compared to the original problem instances. Descent is used as the basis for this comparison, because its simplicity makes it easier to analyse. This comparison is followed by an examination of the model's behaviour, including some techniques that attempt to improve accuracy.

7.1 Simple Descent

Simple descent, described in section 3.1.1, is probably the simplest heuristic search algorithm. This simplicity makes descent useful as a test bed for assessing Barrier Based Models. A Markov model of descent can be easily derived (see previous chapter) and is very powerful. Many search algorithms are similar to descent, so we would expect that many of the characteristics of descent on the model problems also apply to these similar algorithms.

We make an empirical comparison between descent on model problem instances and the original problem instances. To evaluate the results it is necessary to measure some quantity that can be easily averaged across many runs of a descent algorithm. Cost at a particular time is the obvious measure, and this is what is used in the form of average cost charts. Each of these plots compares the average cost at time t as calculated from the Markov model to an average of the costs gathered from many descents on the original problem instance. The average costs of descent on the original problem are calculated from an average over 10000 separate descent runs. The first

500 iterations are compared, as after this time the vast majority of descents have reached a local minimum and can make no further progress. These are compared to Barrier Based models of the same problem instances, using level-accessible sets to partition the landscape (level-connected sets will be evaluated in section 7.1.1). From the model of descent defined in chapter 6.2.1, the average cost at time t, can be calculated as

$$\boldsymbol{p}^T(1)\mathbf{D}^t\boldsymbol{e},$$

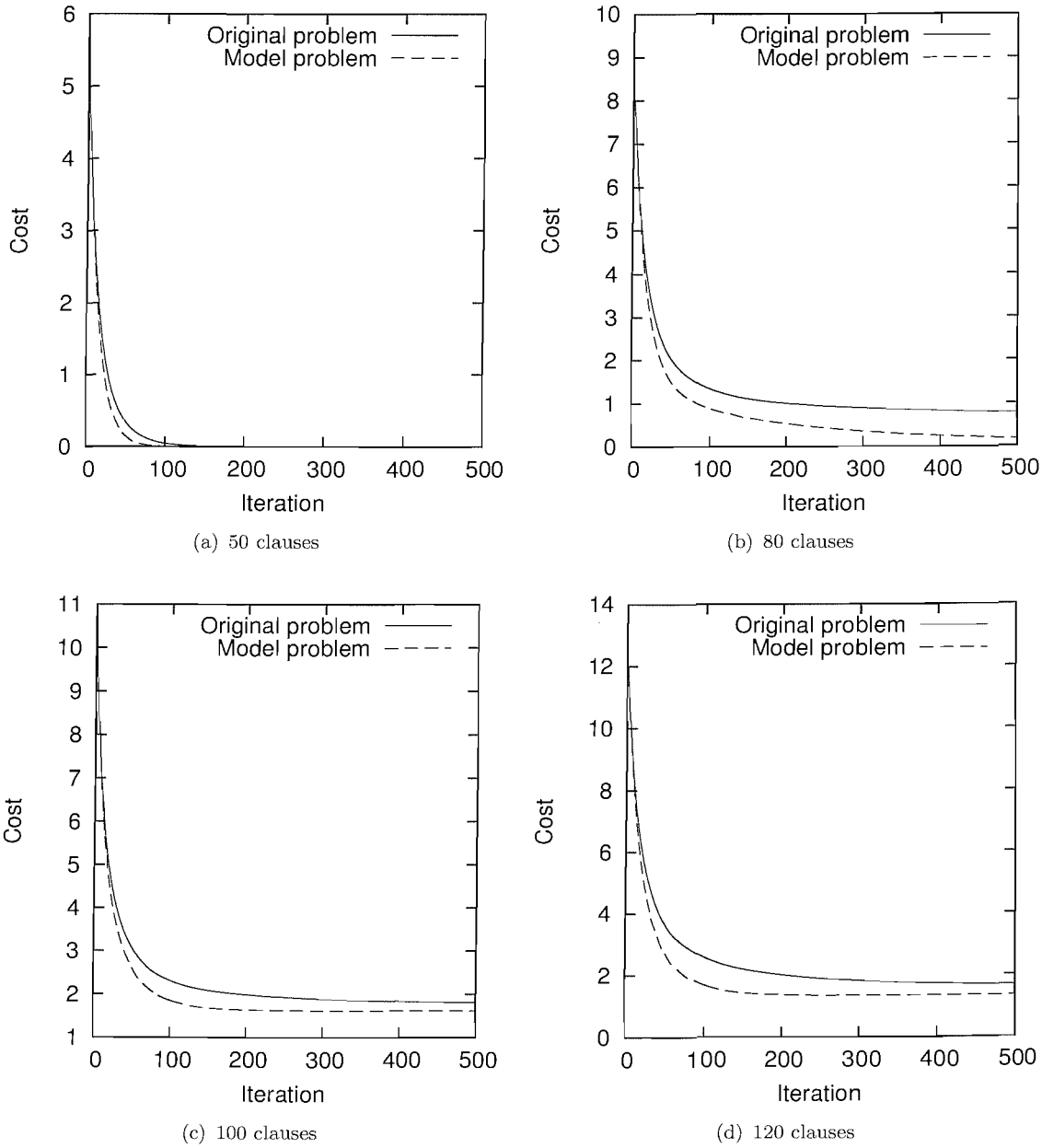
where $\boldsymbol{e}$ is a vector containing the costs of each partition.

The results of these comparisons is shown for Max-3-SAT problems in figure 7.1 and for fully connected Spin-glasses in 7.2. It can be seen that in every case descent is faster on the model problems and almost always reaches lower local minima, on average, compared to descent on the original problem instances. The models of Spin-glass problems appear to match the original problem instances better, however this is explained by the smaller range of costs that the local minima of Spin-glass problems lie over; since most of the Spin-glass instances local minima vary over only a few cost levels, the differences between the descents on the model and the descents on the original instances have less effect on the cost than they do on Max-SAT problems where the minima are spread over a greater range of costs.

The differences in the end cost seem to be systematic. Examining a sample of 50 instances of 20 variable Max-3-SAT problems, in every case the descent on the model instances had a lower average cost after 1000 iterations than on the original problem instances. After 1000 iterations only a tiny fraction (under 1%) of descents have not reached local minima. Figure 7.3 shows the differences in end costs in the form of a histogram.

7.1.1 Level-Connected Set Partitioning

So far only level-accessible sets have been used as the partitioning for our model. As discussed in section 6.2.2, another easily applied partitioning is to use level-connected sets. Level-accessible sets are not necessarily connected, so the model may connect parts of the landscape which are disconnected in the original problem instance. Partitioning the entire landscape using level-connected sets results in a huge number of states, so as a compromise measure, the landscape with a cost lower than the highest cost saddle point is partitioned using level-connected sets, and at this cost and above level-accessible sets are used. Very little time is spent above this cost, and it results in a massive reduction in states, since above the highest cost saddle point there is one level-accessible set per cost level. This makes a Markov chain analysis practicable.



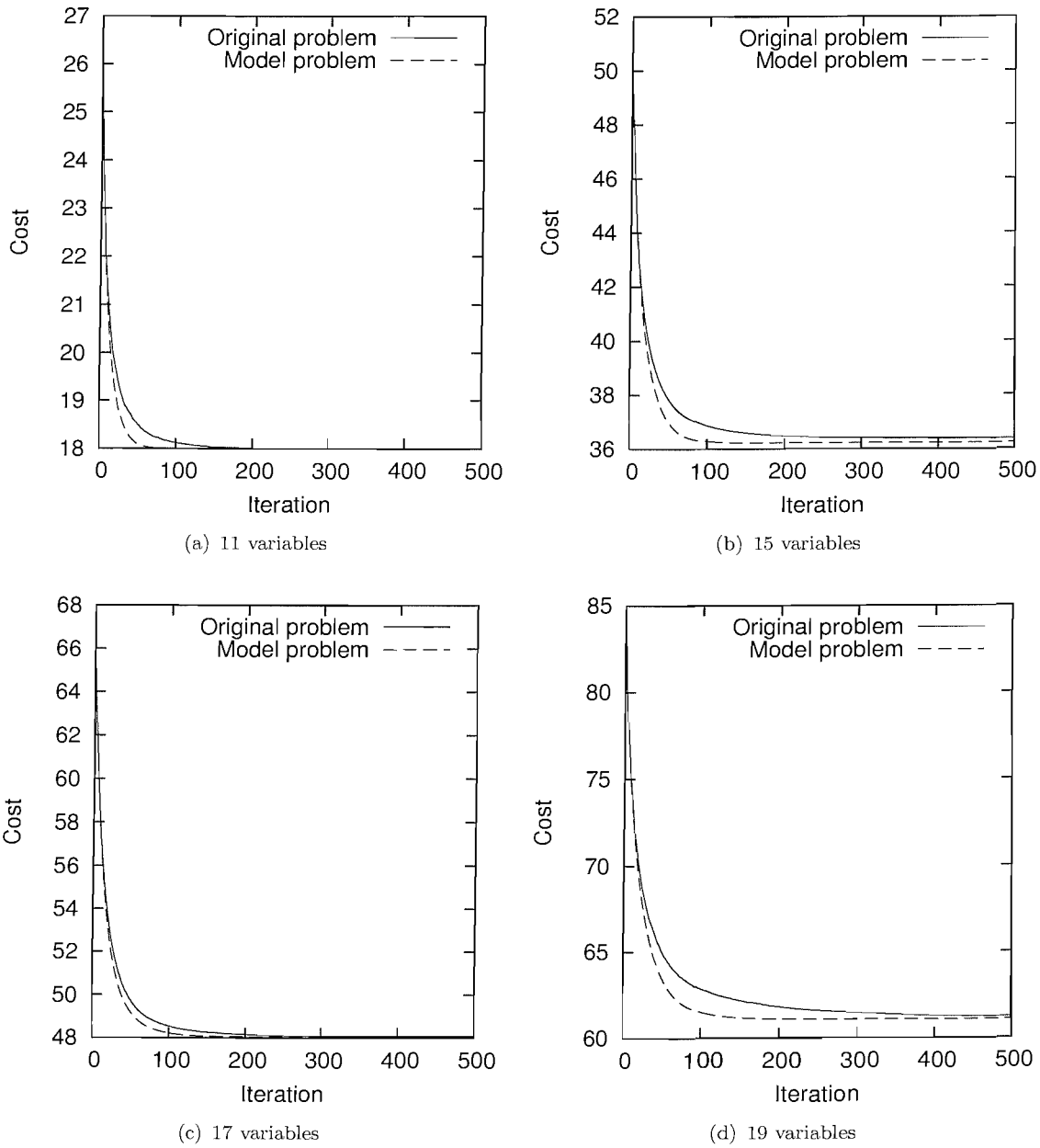
(a) 50 clauses

(b) 80 clauses

(c) 100 clauses

(d) 120 clauses

FIGURE 7.1: Average costs graphs of 20 variable MAX-3-SAT problems



(a) 11 variables

(b) 15 variables

(c) 17 variables

(d) 19 variables

FIGURE 7.2: Average cost graphs of fully connected Spin-Glass problems

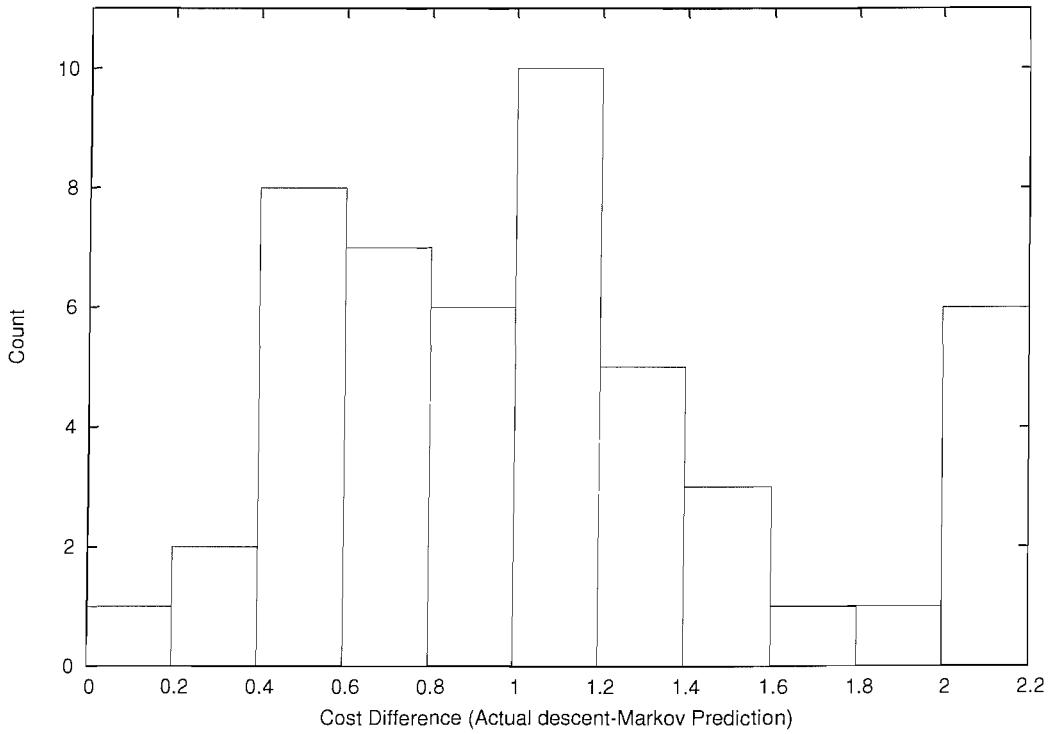
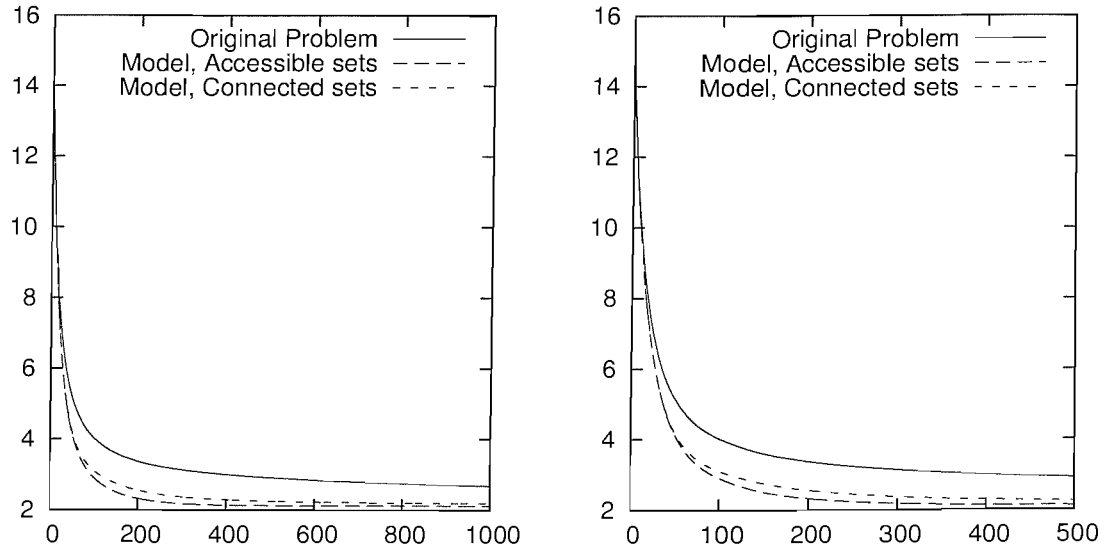


FIGURE 7.3: Histogram of the average difference in cost between model instances and the original instances after 1000 iterations of descent. Sample of fifty 20 variable Max-3-SAT instances.

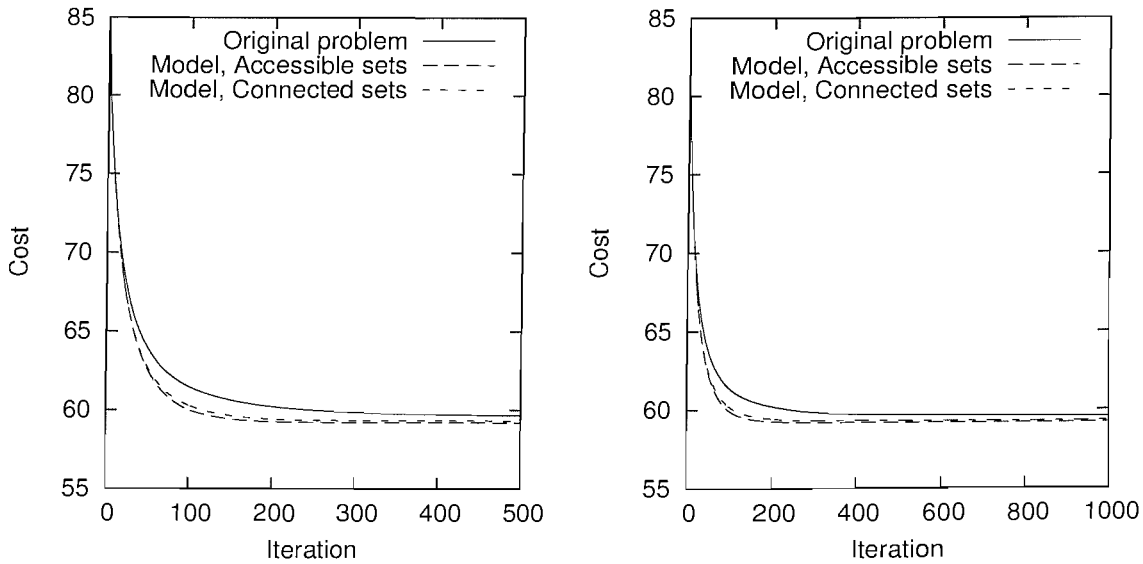
The results using a level-connected set partitioning are shown in figure 7.4. As can be seen, the results using level-connected sets are more accurate. This comes at the cost of a large increase in the number of states however; there are 166 states in level-connected set partitioning compared to 35 in the level-accessible set partitioning. In this small example this causes no problem, but in larger problems such an increase could easily make the Markov chain numerically difficult to analyse. For such a large increase in states, the improved accuracy of level-connected sets is disappointing.

7.1.2 Picking model of Descent

An obvious deficiency of our model is that it does not distinguish between moves to higher cost neighbours and moves to configurations of equal cost. In descent on the real problems, picking a higher cost neighbour means that the algorithm stays in exactly the same configuration, with the same selection of neighbours, while if a neighbour with the same cost is picked the algorithm moves to that configuration and chooses from a different selection of neighbours on the next iteration. This could have a significant affect if the configuration is a 'plateau' configuration, with no lower cost neighbours; many iterations could be spent trying to find an equal cost neighbour to



(a) Max-SAT 20 variables, 80 clauses



(b) Spinglass 19 variables

FIGURE 7.4: Average cost predicted by the model using accessible sets and connected sets (LCS), on a Max-SAT and Spin-glass instances. First 500 iterations displayed separately for greater clarity.

move to without any chance of moving to a lower cost configuration. In our model of descent, the algorithm always gets to choose from all the neighbours of all the configurations in the partition, so there is a probability of moving to a lower cost state in every iteration (assuming the state is not a local minimum).

To reduce this error, and to try gain some measure of its importance, a modified model is tried. We call this the Picking model, and it is described in listing 7.1. Instead of picking neighbours directly based on the probabilities in our connectivity matrix, the probabilities are used to pick members of a 'neighbour list'. Neighbours are then selected from this neighbour list with a uniform probability. The neighbour list is only updated when neighbours with an equal or lower cost are picked; if a higher cost neighbour is selected, the neighbour list does not change. One effect of this is it allows descent to become trapped in a plateau state, as described above.

Pseudo code for the picking model is given in listing 7.1. The function select() returns a number chosen at random with a probability distribution described by the vector passed to it. To introduce some of the cost of moving across a partition, we create a list of neighbours. Each member of this list is picked with a probability from $\mathbf{T}$. There is one restriction upon this list; at least one member must have an equal or lower cost than the current state. This prevents the creation of false local minima. This is achieved by placing the current state as the first member in the list, and the first time it is subsequently picked at random is ignored, and a state is picked again. Neighbours are now picked with equal probability from this list, and the list only gets changed if a neighbour with an equal or lower cost is chosen. This is an iterative model, so is considerably less useful than the simple Markov chain model. Its use is more to try and measure how much of an effect this error in the model has.

Figure 7.5 shows the results of the picking model. The Picking model does have a slightly slower descent than the descent model, but is still faster than descent on the original problem. Only the speed of descent is changed; the proportion that each run finishes in each local minimum is the same.

7.2 Coarse Graining

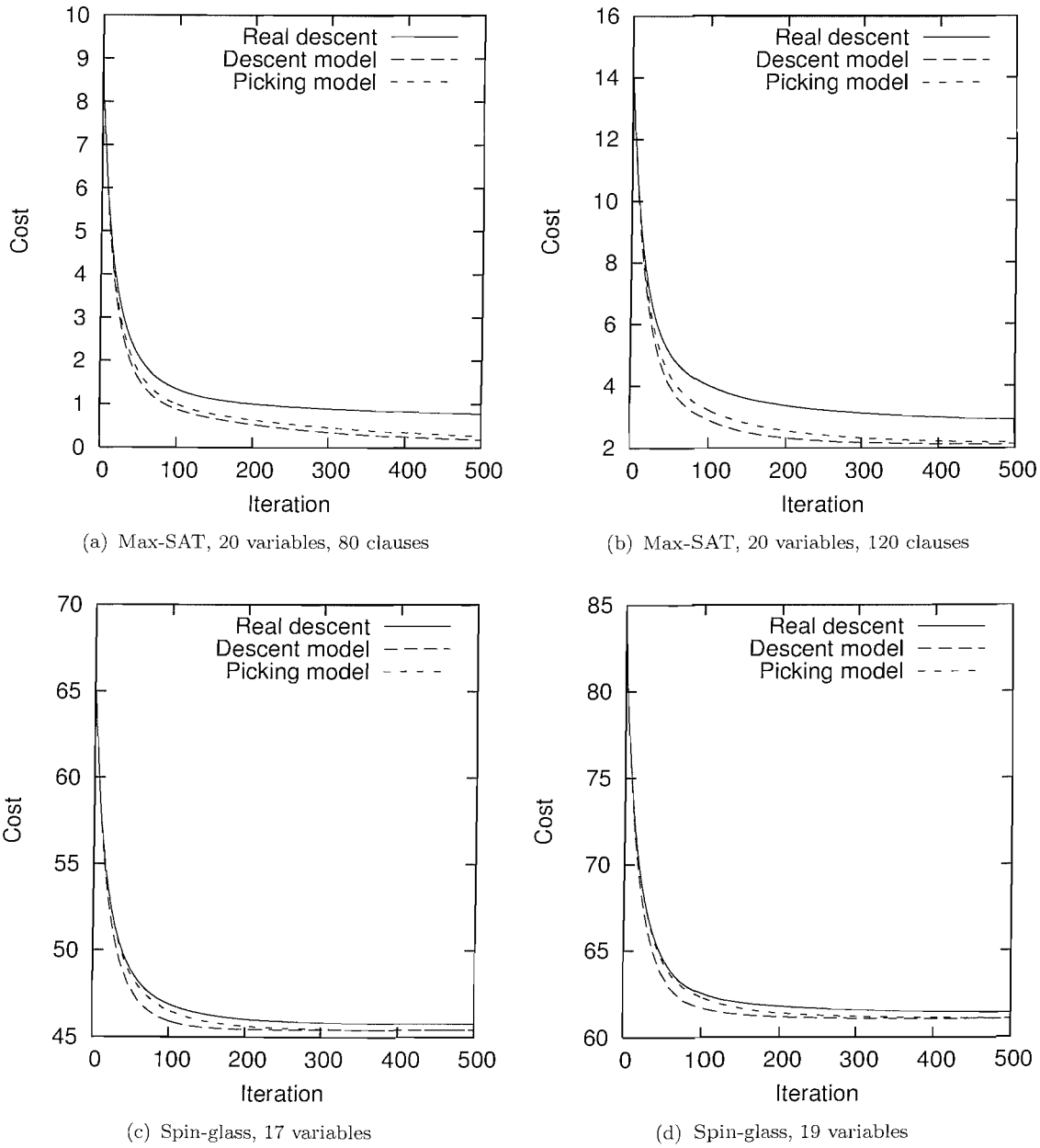
The predictive accuracy of Barrier Based models is not critically important to their use as a study tool. However, it is important that the reasons why and how the model differs from the original problem instances are understood. One of the main advantages of these models over other model problems used to study combinatorial optimisation is that Barrier Based models have a direct analysable relationship to problems representative of those faced by practitioners. In this section we examine

```
s ← select(p(1));
N ← make_neighbours(s, M);

while unfinished()
   //pick from list with uniform distribution
   new=pickFromList(N);
   if f(new) < f(s) then
      s ← new;
      N ← make_neighbours(s, M);
   else if f(new) = f(s) then
      N ← make_neighbours(s); //change list
   endif
end

function make_neighbours(s, P)
   ret ← {s}; //guarantee s is in list
   guard ← true;
   for i=1 to n //making n neighbours
      w ← select(P_s); //selecting from single row
      if w == s ∧ guard then
         //have forced s into list, so first time
         // picked at random re-choose
         guard ← false;
         w ← select(P_s);
      endif
      ret ← insert(ret, w);
   endfor
   return ret
```

LISTING 7.1: Picking model of descent



(a) Max-SAT, 20 variables, 80 clauses

(b) Max-SAT, 20 variables, 120 clauses

(c) Spin-glass, 17 variables

(d) Spin-glass, 19 variables

FIGURE 7.5: Average cost of Descent, with the Descent model and the Picking Model, for various problem instances.

how descent differs on the model problems compared to the original problems. Since many algorithms are very similar to descent (e.g. simulated annealing), it is expected that these results are important to many other types of algorithm as well.

Descent can be seen as a Markov chain on the original problem landscape, with every configuration being a different state. Descent on the Barrier Based models can therefore be seen as a form of coarse graining, where a number of states are collected together as an approximation. The most questionable assumption made in doing this is the assumption that the neighbours of every configuration are as likely to be picked as the neighbours of any other configuration. This is only true if in descent, every configuration within a partition is visited with equal frequency.

This can be easily measured. A count is taken of how many iterations are spent in each configuration of one level-connected set over 10000 runs of descent, each run lasting for 1000 iterations. A 20 variable Max-3-SAT problem is used, and the level-connected set is part of a saddle-point, as the behaviour within local minima has no influence on the outcome. As it is part of a saddle-point, it has a relatively low cost (a cost of 5 when the global optimum is 3 unsatisfied constraints). If the logarithm of the frequency each configuration is visited is taken, the distribution is approximately a normal distribution, as shown in figure 7.6. Table 7.1 shows an analysis of the visiting distributions of three level connected sets from three different problem instances. The distributions were calculated as described above. The Anderson-Darling (Moore, 1986; Stephens, 1974) test compares two distributions and calculates a coefficient that is higher the greater the similarity between the distributions. This coefficient is then compared to the results of distributions produced by a Monte-Carlo procedure, resulting in a p-value. The p-value is the probability, given that the distributions do match, that the coefficient would be lower than the observed coefficient. Therefore, a low p-value rejects the hypothesis that the distributions are equal. As can be seen, the p-values for these level connected sets are all over 0.05, indicating that the hypothesis that these distributions are log normal cannot be rejected on this basis. A log normal distribution indicates that some configurations are visited many orders of magnitude more frequently than others. This is a surprising result, considering the very random nature of descent.

This result is more surprising if descent within each level-connected set is considered as a Markov chain of a closed system, so that descent cannot leave to lower cost configurations. In this case the probabilities are balanced, so that the probability of any transition is equal to the reverse transition; the transition matrix is symmetric (see figure 7.7). In this special case, the stationary distribution is uniform. So to obtain such a significantly non-uniform distribution as a log normal distribution, two criteria must be met for the level-connected set:

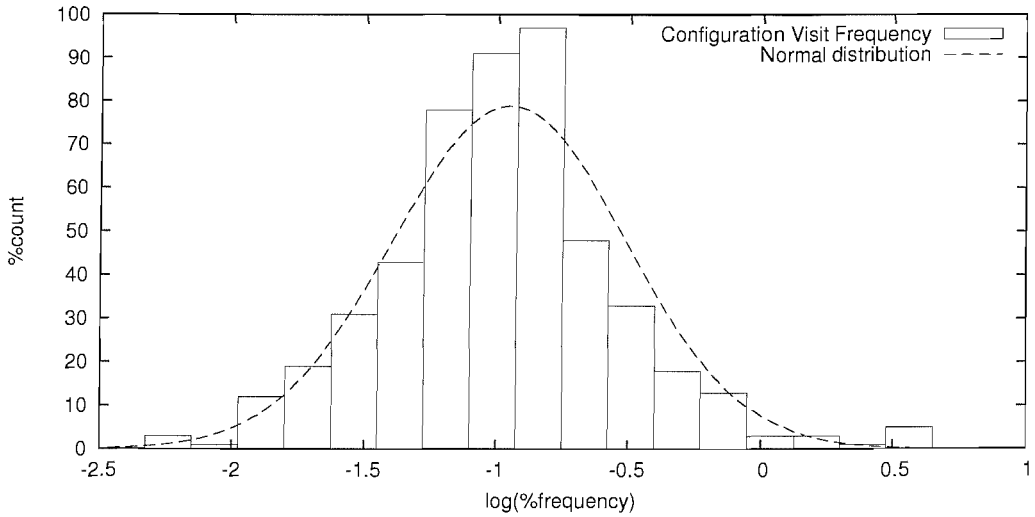


FIGURE 7.6: Log (base 10) frequency that individual configurations within one level-connected set are visited averaged over 10000 descent runs, with a normal distribution with the same mean and variance drawn for comparison.

Problem instance	LCS size	A-D coefficient	p-value
Max-SAT, N=20, 90 clauses	14	0.4235	0.3193
Max-SAT, N=20, 90 clauses	26	0.5257	0.1803
Spin-glass, N=17	5	0.1972	0.8883

TABLE 7.1: Anderson-Darling tests on the distribution that each configuration within a level-connected are visited by a descent algorithm. The p-values for each distribution are all above a significance level of 5%, meaning that our hypothesis that the distribution is log normal should not be rejected.

1. The starting distribution, that is the configurations at which descent enters a level-connected set, must be non-uniform.

2. Descent must leave each level-connected set before significant mixing can occur.

While the second point may not seem very likely, the level-connected sets we are looking at are low cost, so exits to lower cost configurations are rare, and leaving does take some time. In addition, very little mixing would be enough to disrupt the log normal distribution observed.

It is possible to directly calculate a measure of the mixing time for each level-connected set, as described in section 6.3.1. It is also possible to calculate the mean exit time, by collecting all lower cost configurations that neighbour the state into a single absorbing exit state (see figure 7.8), and then calculating the mean first passage time to that state, as described in section 6.3.1. The mean first passage time calculation requires a starting distribution; we use a uniform distribution, and a distribution calculated empirically from many descent runs. The values for several

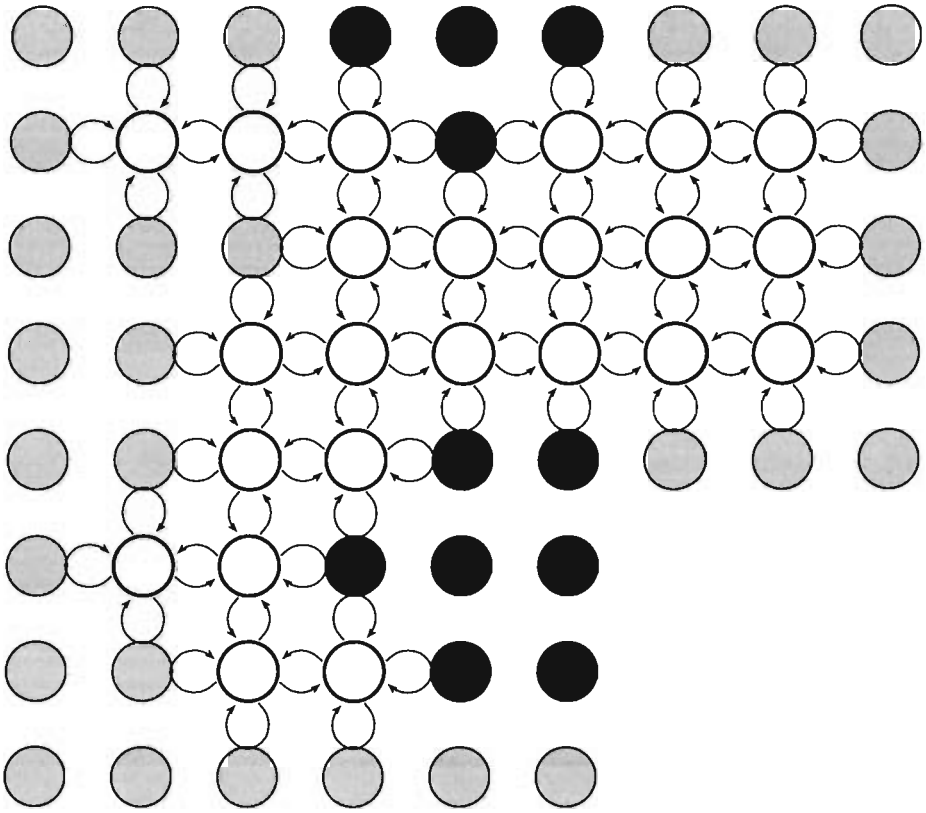


FIGURE 7.7: Representation of a (simplified) partition as a Markov chain. The white circles represent configurations within the partition, the grey circles configurations with a higher cost and the black circles configurations with a lower cost. In this case we are treating the partition as 'closed', so the process cannot enter lower costs states as well as not being able to enter higher cost states. In this case, the probability of entering a state is equal to the probability of leaving a state, so the probabilities are balanced and the steady-state distribution is uniform.

level-connected sets from a 20 variable Max-3-SAT problem with 120 clauses are presented in table 7.2. Only the larger level-connected sets from the joining nodes of the tree are shown (i.e. not local minima). The Barrier Tree of the instance from which these level-connected sets are taken from is shown in figure 7.9; the internal nodes are labelled so that it can be seen from which part of the tree each level-connected set comes.

The exit times cannot be compared directly to the value given for the mixing time; the mixing time is only a measure, and the actual time to mix depends on other factors which are not taken into account. However it can be seen that the mixing time is often one or two orders of magnitude larger than the exit times, especially with the larger level-connected sets A1, B1 and C1. This indicates that very little mixing is occurring. It can also be seen that the mean exit time is almost always longer for the empirical data than the uniform distribution. The only exception to this is the level connected set E1. Descent's distribution of entry into a level-connected set is

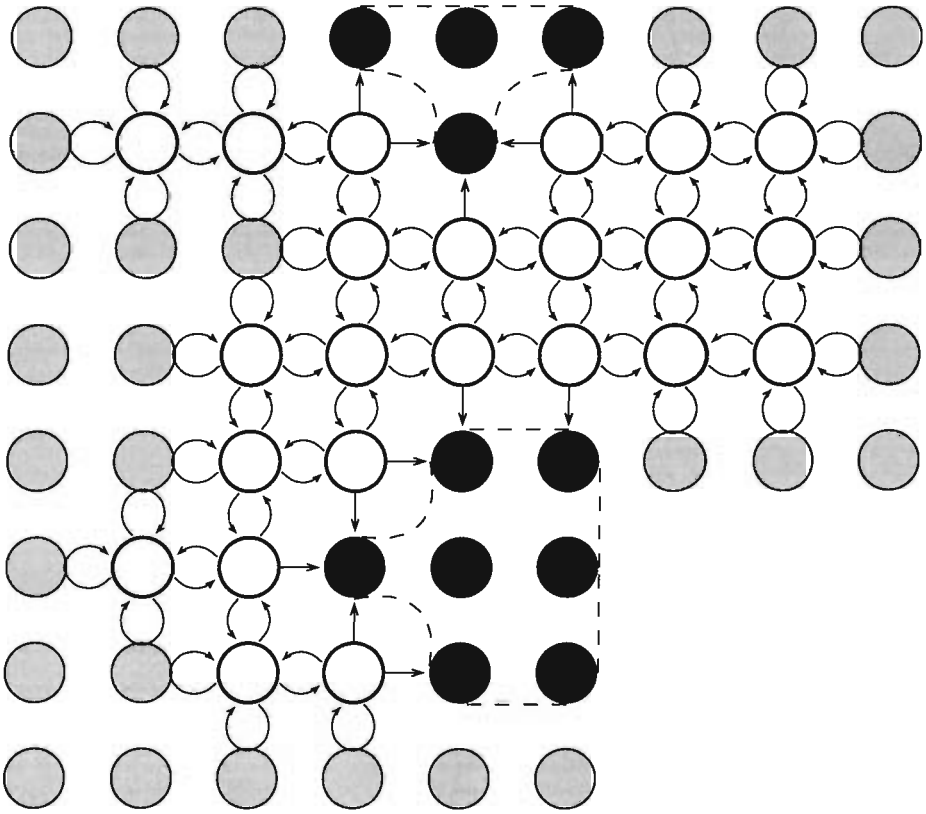


FIGURE 7.8: The same Markov model as in figure 7.7, except the lower cost configurations, in black, are considered to be a single absorbing exit state. The two separate groups, while unconnected, are represented by the same Markov state.

Connected set	Cost	Size	Mixing	Exit (Uniform)	Exit (Empirical)
A1	7	4493	2303	11.87	15.966
A2	7	21	212.0	7.473	8.3886
A3	7	19	209.9	148.4	291.89
B1	6	1642	2562	19.21	27.364
B2	6	14	48.92	18.06	18.310
C1	5	341	2480	40.09	50.731
C2	5	11	47.03	246.7	263.47
D1	5	5	21.62	22.32	22.682
E1	4	14	236.1	70.48	66.829

TABLE 7.2: The mixing and exit times for the larger level-connected sets that are part of the saddle-points of a Max-3-SAT problem with 20 variables and 120 clauses.

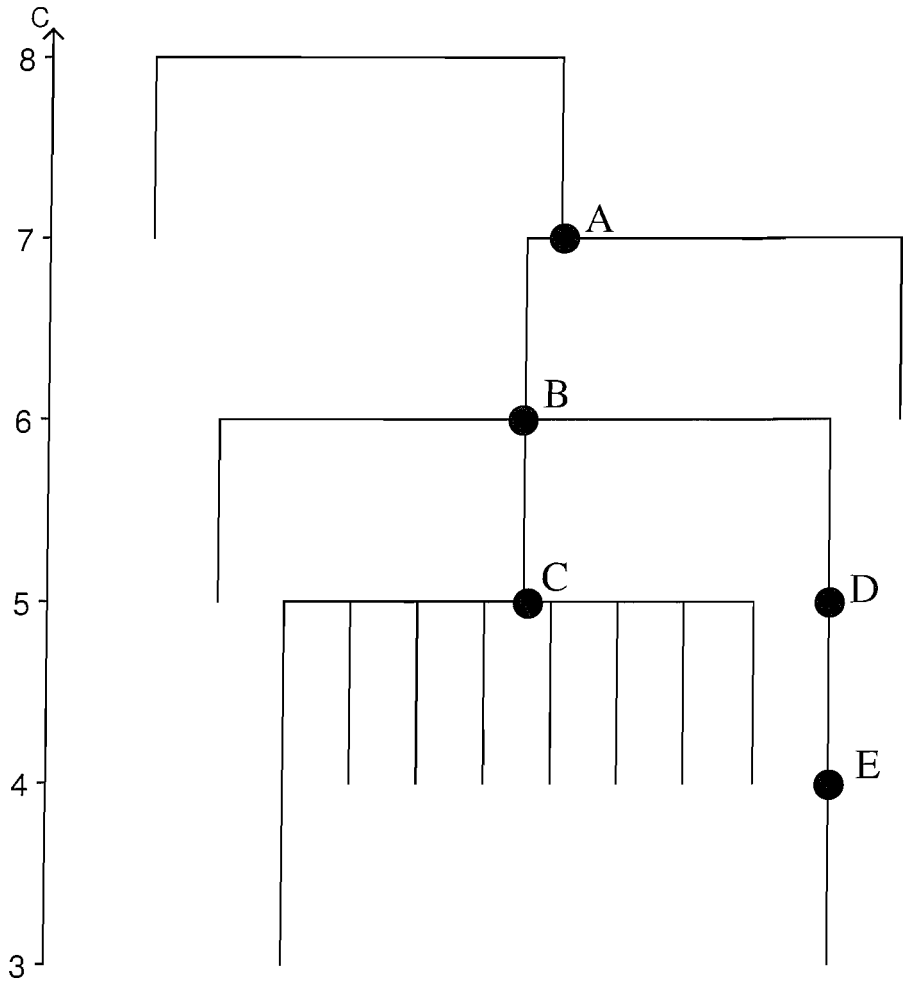
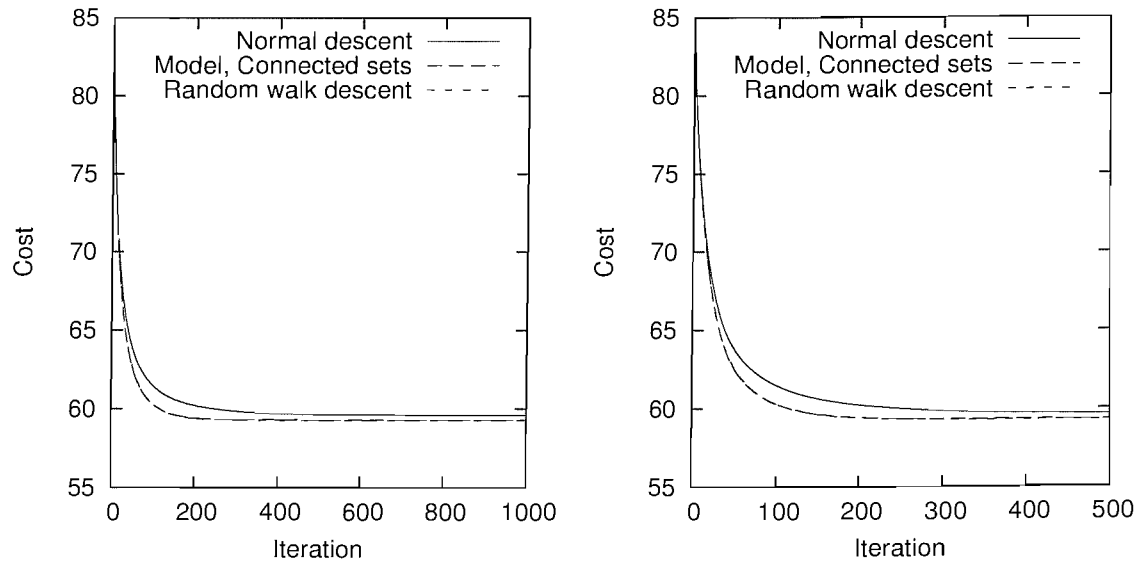


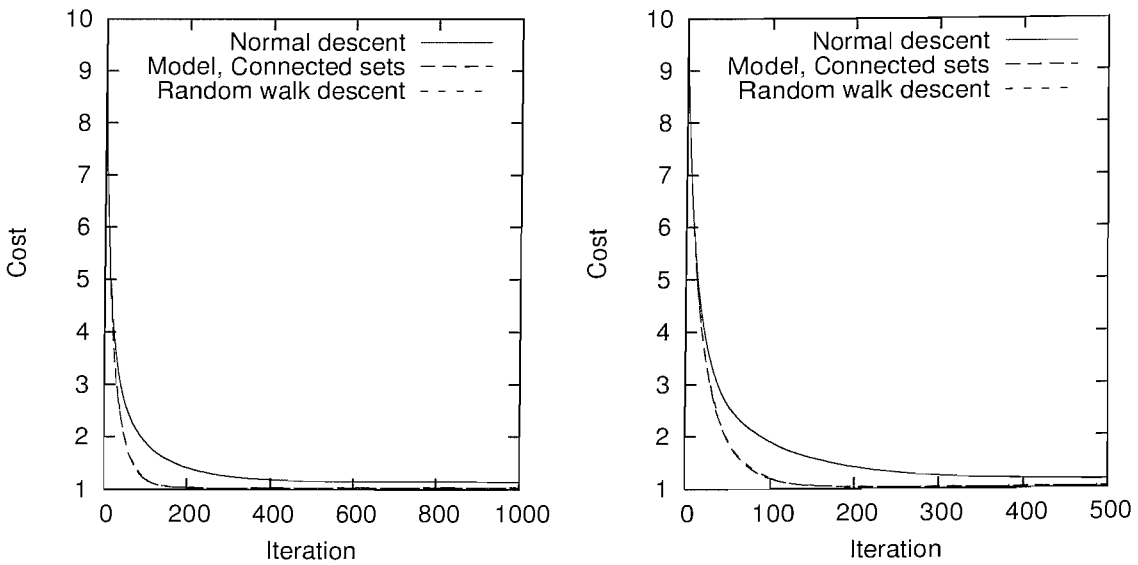
FIGURE 7.9: Barrier Tree of a 20 variable Max-3-SAT problem with 120 clauses.

skewed so that most entries are a greater than average distance from the exit. It is an open question as to why this should be the case.

As a final demonstration of the effect mixing has on descent and our models, a special version of descent acting on the original problem instance can be run. This version of descent is modified to guarantee proper mixing; after a move which lowers cost, the algorithm performs a random walk for 5000 attempted steps, where it is not allowed to move to configurations with a lower cost as well as to higher costs. The random walk is therefore constrained to be within a single level-connected set, and thus mixes within that level-connected set, creating a uniform visiting distribution. The results of such a descent can be seen in figure 7.10, and it can be seen it matches the model descent with a level-connected set partitioning very closely. This is also useful verification that the tools used to build the models are working correctly.



(a) Spin-glass, 19 variables



(b) Max-3-SAT, 20 variables, 80 clauses

FIGURE 7.10: Average cost of a descent algorithm with a random walk, restricted to configurations of the same cost and a length of 5000 attempted moves, for an instance of a Spin-glass and a Max-3-SAT problem. The line for the model descent and for the random walk descent are on top of each other; the model using connected sets matches the descent with random walk very closely

We have provided a breakdown of the cause for the difference between descent on the model and on the original problem, at least in the case of the Max-3-SAT problem. It has been shown that the difference is caused by the assumption that descent visits every configuration within a level-connected set equally frequently, which would be the case if descent entered level-connected sets with a uniform distribution or spent long enough in each level-connected set to mix thoroughly. While we have shown that neither of these is the case, the question of why configurations within a level-connected set are visited with a log normal distribution is unanswered. It also raises the question of how the models could be altered to provide a better fit to the true problems, something that is touched on in the next chapter.

Chapter 8

Limitations and Improvements to Barrier Based Models

Barrier Based models can be applied to a wide range of problem classes and are a powerful analytic tool. This chapter looks at some of the limitations of the models, and some possible methods of pushing back these limits. Firstly, ways of further partitioning the landscape to improve the accuracy of modelled descent are explored. Secondly, the difficulty in using the model with complex search operators that do not have a simple neighbourhood is looked at, and a specific method of modelling crossover is described and evaluated. Thirdly, the limits of size and complexity of problem instances that can be modelled are discussed, and it is shown how sampling can be used to model far larger problem instances.

8.1 Mixing Problem and Accuracy

As shown in chapter 7, descent on Barrier based models differs from descent on the original problem instance. This inaccuracy arises because of the uneven frequency with which configurations within the partitions are visited. While this inaccuracy does not invalidate the use of the model as an analytical tool, greater predictive accuracy would provide greater confidence in the model.

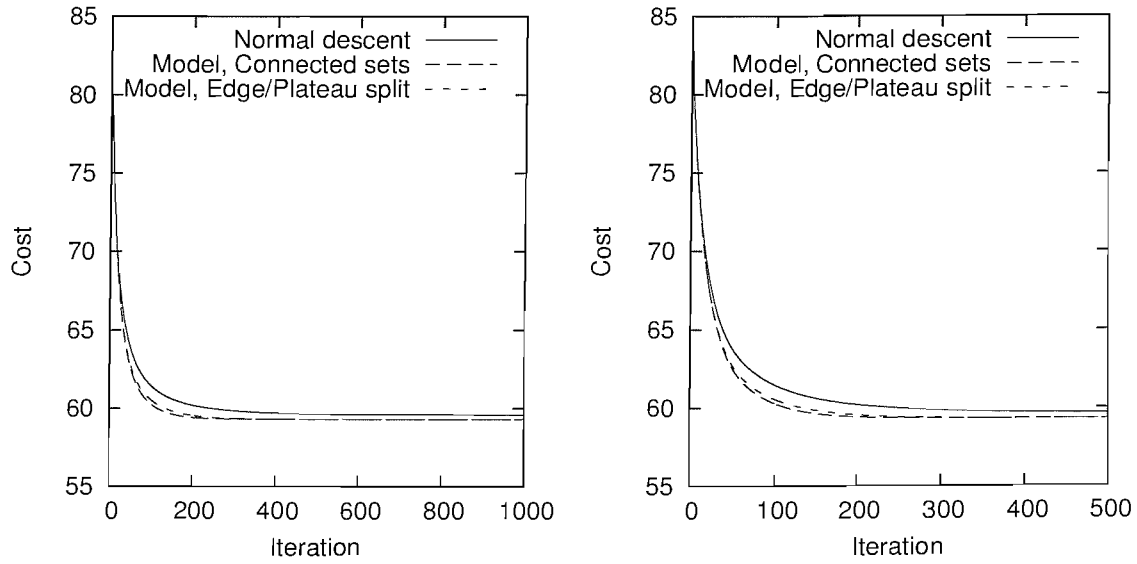
One approach to improve the accuracy is to try and devise a way of calculating the transition matrix so that it is more accurate, such as generating the probabilities empirically from descent. However, such a transition matrix would apply only to descent, and could not be adapted to other search heuristics easily. A different way of improving the accuracy is to change the partitioning. This has the advantage that

it does not change the fundamental model, and it is clear that some partitionings must be more accurate than others; if a partitioning with one configuration in each state is used, the model of descent is exact. Introducing more states into the model comes at the cost of a larger model that is more difficult to analyse; however for the smaller problem instances looked at so far, the models have been small enough that the number of states could be increased by an order of magnitude and still have a useful model.

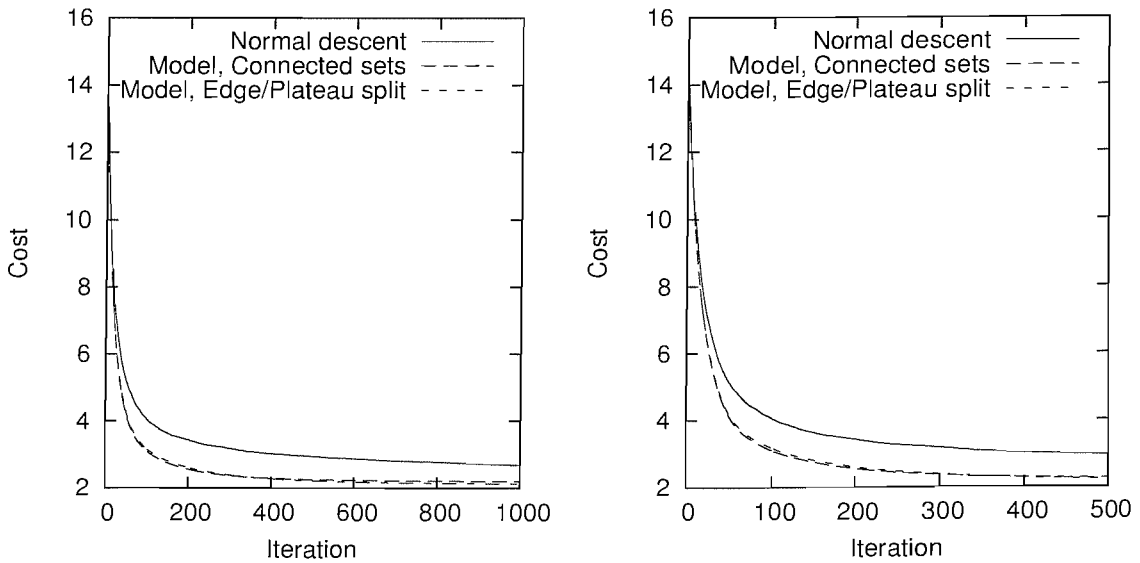
The approach considered in this section is to further partition level-connected sets into edge and plateau sets. The edge set of a partition consists of all the configurations that have a lower cost neighbour. The edge configurations are important, since descent algorithms must pass through these configurations to reach lower cost configurations. One example of the inaccuracies related to these edge states is that with our normal level-connected set partitioning, descent can exit a partition on the first iteration after entering that state. In the original problem, a descent algorithm will often have to pass through several states before it can reach a lower cost configuration. Separating out the edge partitions ensures that descent must spend at least one iteration moving to a configuration with lower cost neighbours, and the expected time to descend to a lower cost state is increased to something closer to the original problem. Another advantage of splitting level connected sets in this fashion is that it increases the number of states by a limited amount; one extra state per connected-set that isn't a local minimum is created.

The results of using this partitioning are shown in figure 8.1. Descent on the model using the edge partitions is slightly slower and therefore slightly closer to descent on the original problem. The improvement is small, and the average end cost is almost unchanged. The increase in the number of states is also small; 112 with the edge splitting compared to 90 without, in the case of the Max-3-SAT problem.

Finally, another adjustment is made to this partitioning. In the above partitioning, all configurations with lower cost neighbours were put in an edge state, no matter where those lower cost configurations are in the landscape. As a further partitioning, the edge partitions are split depending on the lower cost level connected set in which they have neighbours. A single state is created for all the configurations within a level connected set that have lower cost neighbours in several different level-connected sets. The results of this partitioning are shown in figure 8.2. It can be seen that this offers only a very small improvement over not splitting the edge states. This partitioning also creates a large number of extra states; 274 states compared to 112 just splitting the edges and partitions in the case of the Max-3-SAT problem shown. Such a small improvement for so many more states makes this partitioning less useful.

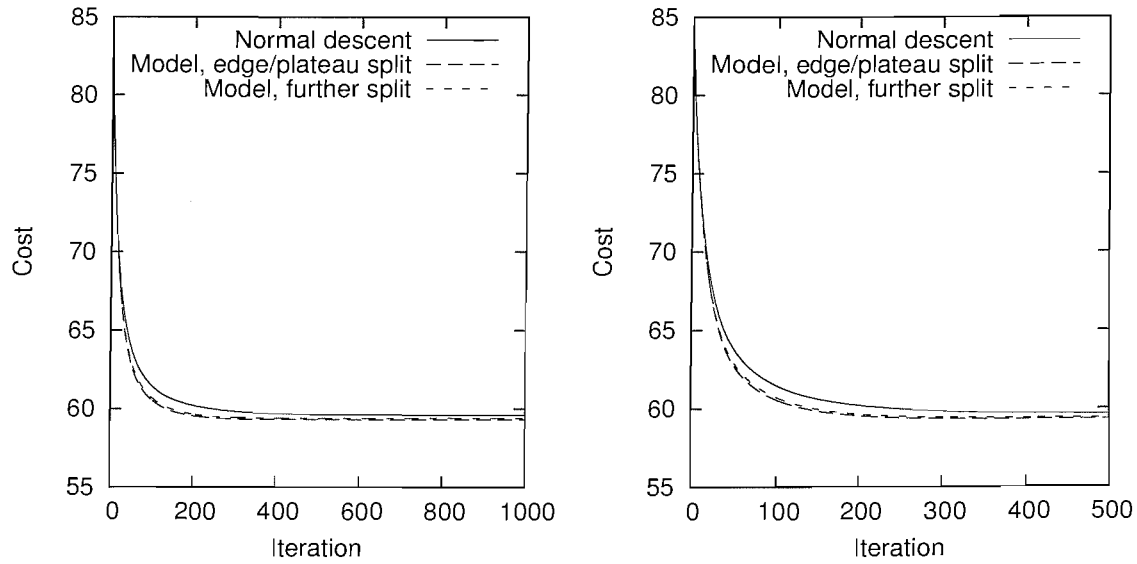


(a) Spin-glass, 19 variables

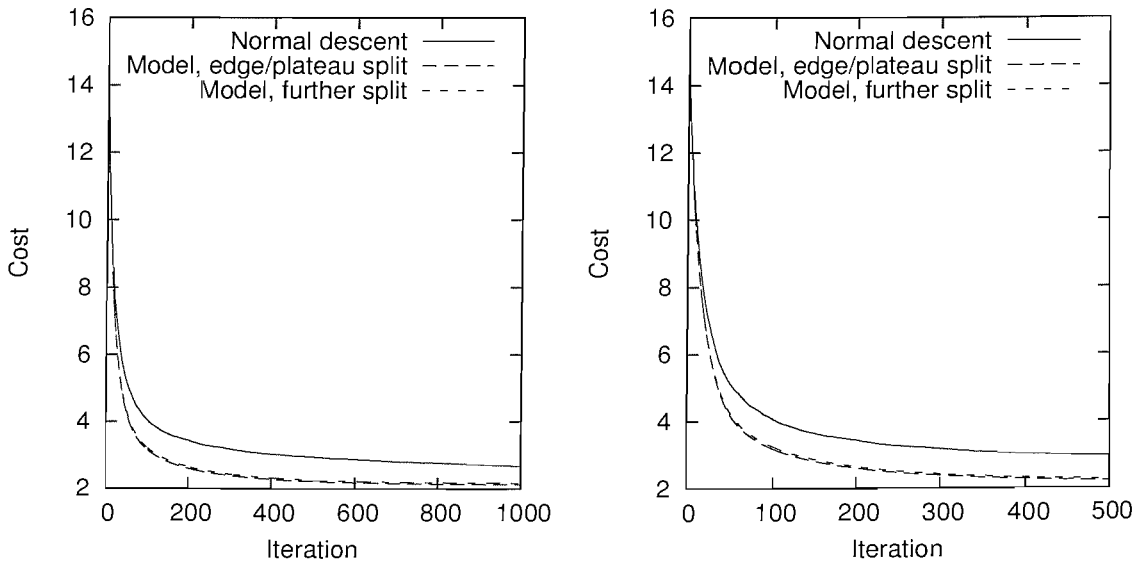


(b) Max-3-SAT, 20 variables, 120 clauses

FIGURE 8.1: Comparison of models using a level-connected set partitioning, and a level-connected set partitioning with each partition further split into edge and partition sets.



(a) Spin-glass, 19 variables



(b) Max-3-SAT, 20 variables, 120 clauses

FIGURE 8.2: Comparison of descent on model problems using a plateau/edge partitioning and further partitioning the edge states into which level connected set they lead to.

8.2 Limits of the Neighbourhood Model and Crossover

Complex search operators such as crossover do not have a simple neighbourhood, as discussed in chapter 3. So far, Barrier based models have assumed a simple neighbourhood, so crossover and other complex operators cannot be easily represented. In this section one way of representing a particular type of crossover is explored.

It should be realised that the difficulty of representing crossover is only one of the challenges of modelling Genetic algorithms in this framework. The use of a population creates an immediate difficulty in any Markov model, as the number of states necessary rises exponentially in the size of the population. There are some well established techniques for dealing with some of these problems, such as the infinite population model (e.g. Vose (1999)). This section should be seen more as how the challenge of modelling complex operators can be approached in this framework, rather than a practical guide to modelling a Genetic algorithm.

The type of crossover looked at in this section is uniform crossover between two binary strings. This is one of the most popular forms of crossover. This cannot be represented directly by our connectivity matrix, because the the key information used in crossover, the binary string representation of configurations, is removed in the model. Our connectivity matrix assumes a simple neighbourhood, and creating a connectivity matrix that reflects the neighbourhood of uniform crossover is challenging (for example, see Stadler and Wagner (1998)). Instead, we look at a method of modelling uniform crossover using the basic connectivity matrix based on the Hamming neighbourhood.

To do this, it is necessary to introduce some additional information about the original problem instance into the model. The information added in the model of crossover presented here is the average Hamming distance between the different partitions. This can be efficiently calculated from the correlation, which is described in section 5.2.

Crossover is modelled in the following way. The connectivity matrix allows the result of any number of random bit flips to be predicted. The number of bit flips between the result of uniform crossover and one of its parents is dependent on how many bits are different between the two parents (i.e. the Hamming distance). If two parents differ at z sites then the probability of a child differing from one of the parents at n sites is given by a binomial distribution (at each of the z sites there is a probability of one half that the child inherits the allele from that parent). Formally, given two states $\mathcal{P}_a$ and $\mathcal{P}_b$, with an average Hamming distance z then the probability of the

Rank	Real Crossover %	Predicted % for same state
1	20.4	1.58
2	14.4	4.31
3	12.5	28.2
4	12.4	1.58
5	11.8	19.3
6	11.3	24.5
7	7.98	13.1
8	4.69	5.2
9	2.26	1.68
10	1.05	0.426

TABLE 8.1: The top 10 most frequent results of crossover performed, compared to the prediction from our crossover model. The parents of the crossover were chosen from the states at rank 1 and rank 4.

child ending up in state $\mathcal{P}_c$ is taken to be

$$X_c(a,b) = \frac{1}{2^z}\sum_{n=0}^{z}\binom{z}{n}\frac{P(c|a,n)\,P(c|b,z-n)}{\sum_{c'=1}^{N}P(c'|a,n)\,P(c'|b,z-n)}$$

where $P(c|a,n)$ is the probability of moving to state $\mathcal{P}_c$ from state $\mathcal{P}_a$ in n steps. That is,

$$P(c|a,n) = \boldsymbol{\delta}_a^T \mathbf{M}^n \boldsymbol{\delta}_c$$

where $\boldsymbol{\delta}_a$ is the unit vector with 1 in the state $\mathcal{P}_a$.

8.2.1 Results

To evaluate this model of crossover, two level-connected sets from a single instance of a 20 variable Max-3-SAT with 100 clauses are used. The two level-connected sets were chosen at random with the restrictions that they contain more than 20 configurations but less than 100.

Table 8.1 shows the top 10 most frequent states for the result of crossover to end up in, and compares them to the prediction from our simple model. The states that the parents were randomly chosen from are at rank 1 and rank 4. As can be seen, our model's predictions are poor.

In conclusion, it has been shown that a plausible model can be derived for a complex search operator such as uniform crossover using the simple connectivity matrix based upon the Hamming neighbourhood. While this model performs very poorly in accurately predicting the result of crossover, it should be remembered that this is a simple adaptation of the model designed for the simple Hamming neighbourhood. A

more sophisticated adaptation, or a more appropriate connectivity matrix should be able to produce a more accurate prediction.

8.3 Size Limitations and Larger Models

The problems that can be represented as Barrier Based models are "real" problems in the sense that they come from problem classes that are exempla of the hard problems faced by practitioners. The actual instances that can be modelled are so small that finding the solution is computationally trivial; although the models themselves are still interesting. There are two main limits on the complexity of problems that can be modelled.

Firstly, the size of the cost landscape. To get the full structure of the Barrier Tree, we need to evaluate the cost of all configurations beneath the highest cost saddle point, and to calculate transition probabilities for the higher cost states, we need to evaluate a significant proportion of configurations at each cost. So far, we have assumed that the entire landscape has been evaluated, which limits us to problems with around 8 million configurations, or 23 binary variables. Using sampling techniques, it is no longer necessary to completely evaluate the entire cost landscape, and this allows far larger problem instances to be modelled. This will be described in this section.

The second limit on the complexity of problems that can be handled is the size of the resulting model. If there are too many partitions, performing Markov chain calculations becomes impracticable. This is a somewhat flexible limit, as it depends largely on what calculations are desired. The number of sets required to partition the landscape is influenced by several factors, including the problem class, size and complexity. The main variable in the cost landscape that affects the number of states in the model is the number of local minima; each local minimum corresponds to one extra state in the model. This varies between problem classes and the parameters controlling the problem instance generation. For example, Binary perceptron problems tend to have a large number of local minima, so many that this is the main limit on the size of instance we can model. With Max-SAT, the number of clauses per variable increases the number of local minima, and also increases the range of costs in the landscape, both of which increase the number of states required by our model.

A related issue is the range of cost values in the problem instance. So far, only problems with highly degenerate landscapes have been looked at. Other problems, such as the Travelling Salesman problem, have a far greater range of costs, so much larger that every configuration may have a unique cost. The partitionings that have

been looked at always separate configurations with different costs. For problems with a large range of cost values, this will lead to a correspondingly large number of partitions. In practice, a different partitioning will need to be devised to make modelling problems with a large range of costs practicable.

Sampling

Previously, it has been assumed that the entire landscape has been exhaustively enumerated. This allows the size and neighbourhood connections of each partition to be counted exactly. As previously described, the Barrier Tree of the landscape has a trivial structure for all of the landscape above the lowest cost saddle point, and there is no need to evaluate the configurations with a higher cost to calculate the Barrier Tree (see 4.3). In a similar fashion, it is not necessary to enumerate these configurations to calculate a Barrier Based model. The low cost configurations can be enumerated and the model calculated exactly for them. For the higher cost configurations, the landscape can be randomly sampled to generate a good approximation of the higher cost partitions. This approach relies on the fact that most search algorithms spend very little time in the higher cost parts of the landscape, so inaccuracies introduced by the sampling process have little effect on the model.

The limit to the size of problems that can be modelled with this technique is difficult to define exactly. As the size of the problem instances gets larger, the landscape can't be sampled as well, leading to greater inaccuracy in this process. It also becomes more difficult to evaluate the lower cost portions of the landscape. Perhaps a less obvious difficulty is that as problems become larger it can become difficult to find a suitable cost boundary at which to cut; with large problems a single cost level can contain many millions of configurations, which can limit the fully evaluated section of the landscape to only a small proportion of what would otherwise be possible.

The method for using sampling to create the model is quite simple. The Barrier tree of the problem instance is calculated, as described in section 4.3. The transition matrix for the partitions below the cut off cost can then be calculated as normal. For the states above the cut off point, it is assumed that they are above the highest cost saddle point, so that every configuration with an equal cost is in the same level-accessible set, and therefore in the same partition. Random samples are then taken, and the neighbours of each sample looked at. The number of samples at each cost level gives an approximation of the proportion of configurations at each cost level, and the neighbours of each sample can be used to approximate the transition probabilities.

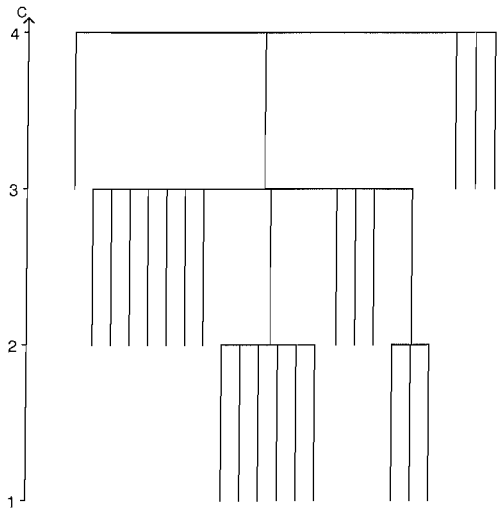


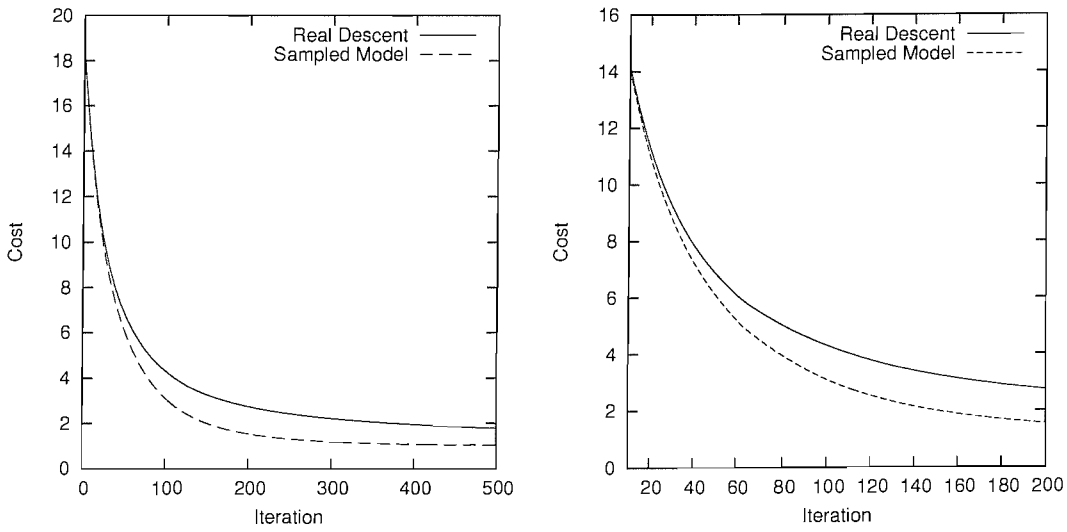
FIGURE 8.3: Barrier Tree of 40 variable Max-3-SAT problem.

8.3.1 Results

We demonstrate these techniques on a 40 variable MAX-3-SAT problem with 160 clauses. The Barrier tree of this instance is shown in figure 8.3. This has a cost landscape of a non-trivial size, having approximately 10^{12} configurations. The cost bound was set so that configurations with a cost greater than 4 are not evaluated, which in this instance gives us 3 million states in the completely evaluated part of the landscape. The rest of the landscape was sampled 1 million times. To evaluate the error introduced by using sampling instead of a complete evaluation of the landscape, we repeat this sampling 3 times so that the results can be compared. The models produced each had 65 states.

Figure 8.4 shows the results, compared to real descents. Only one of the three sampled runs is shown; the three runs were almost identical, with the greatest standard deviation between the cost predicted by any of the models at any iteration being 0.0178. The results compare favourably with smaller, completely enumerated landscapes.

The main risk of not evaluating the whole landscape is that there will be local minima above the root of the tree which will not be part of our model, and as such, our model will not predict descent runs ever getting trapped at a level above the bound. In our example, we ran 1 million descent runs on the original problem, and none of them ended in a local minimum above the root, so any local minima above the root that do exist must be of little significant to descent.



(a) Descent to end state

(b) Initial Descent

FIGURE 8.4: Comparison of a sampled 40 variable MAX-3-SAT problem to real descent.

Chapter 9

Conclusions and Future Work

Barrier trees and Barrier based models are tools to study heuristic search on Combinatorial optimisation problems. However, the focus of my PhD has been the development of the tools, and no time has been spent using them to actually study Combinatorial optimisation. This has been due to lack of time and to a lack of computing resources. The potential of Barrier Trees and Barrier based models to further our understanding of Combinatorial optimisation and heuristic search should be clear. Barrier Trees provide an insight into the form taken by the cost landscapes of challenging problem classes. The characteristics of these landscapes are hugely important to search algorithm design, but are complex and difficult to work with, so are poorly understood. Barrier based models allow search heuristics to be numerically analysed on model problems that are far more interesting than the toy problems typically used for this purpose, and hopefully have a direct relationship to the original problems.

There are already some examples of the usefulness of the tools. Will Benfold has used Barrier based models that I have generated to calculate simulated annealing cooling schedules (Benfold et al., 2005a,b). A gradient descent method is used to optimise the cooling schedule of a Markov chain representation of simulated annealing on Barrier based model problems. The optimised annealing schedule can then be used on the original problem and other problems of the same class. The annealing schedules optimised on individual instances perform relatively poorly, but optimising the schedule across models of several instances does produce a schedule that performs well (see figure 9.1). This suggests that while the Barrier based models may not be very accurate, they can capture characteristics of the problem class that are important to heuristic search algorithms.

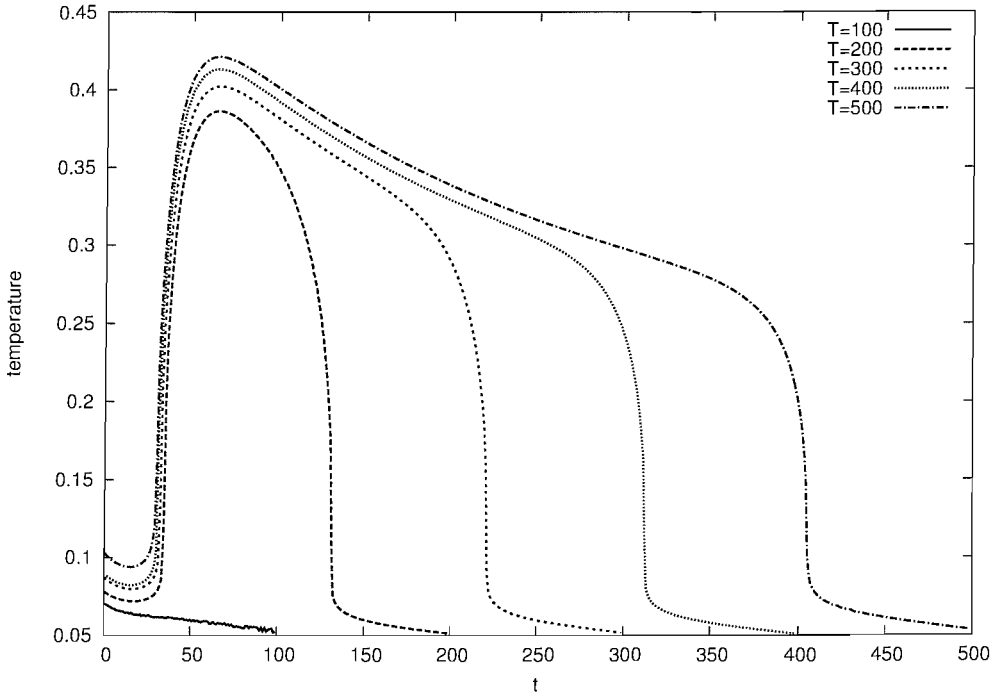


FIGURE 9.1: Optimal annealing schedules calculated using Barrier-based models by Will Benfold. Each line is the schedule for an algorithm lasting T iterations. The schedules are optimised for "where you are", the cost at the finish of the algorithm, as opposed to "best seen", the best solution seen by the algorithm. These schedules are optimised for an instance of Max-SAT.

The usefulness of Barrier Trees to study Combinatorial optimisation had already been independently recognised by other researchers, for example Stadler et al. (2003). It is hoped that the work of this thesis will bring Barrier Trees to the attention of a larger number of researchers in the field. The innovation we introduce of partitioning the entire cost landscape so that every configuration is mapped to the tree greatly increases the power of Barrier Trees. The Barrier Tree can now reveal not just the depth of a basin surrounding a local minimum, but the number of configurations in that basin. Perhaps more interestingly, it also allows the shape of these structures to be studied. This could be very useful; for example, knowing the general shapes of plateaux in the cost landscapes of a particular problem class could allow a Tabu algorithm to be designed that could escape from those plateaux quicker.

One observation that we have already made using these mappings is the observation in chapter 7 that certain configurations within plateaus are visited many orders of magnitude more frequently than others, even by a random descent algorithm. The observation that simple search seems to be channelled into a surprisingly small channel of exploration is one that could be very difficult to observe on larger problem instances, but could be very important to the performance of heuristic search algorithms.

The main weakness of Barrier Trees and Barrier based models as tools is the need for some sort of scaling analysis. It is not known whether the small instances that these tools can be used on share any important characteristics with the large problems that are challenging to modern computers and heuristic search algorithms. Ideally, a scaling analysis would allow Barrier Trees and Barrier based models to be generated with the characteristics of problem instances far too large to generate the trees directly. The fact that Barrier Trees can be generated for cost landscapes varying over many orders of magnitude, from instances with 10 binary variables (approx. 1000 configurations) to 50 variables (approx 10^{15} configurations) should help make a scaling analysis practicable.

Several possible improvements for Barrier Trees have already been given at the end of chapter 5. There are also several improvements that could be made to the Barrier based models. With the breakdown of why the models are inaccurate given in chapter 7, it should be possible to design a partitioning that results in greater accuracy. This idea is touched on in section 8.1 splitting the edges from the plateaux, but with more work it seems likely that a better solution could be found.

Barrier Trees and Barrier based models are well developed tools. The most important future work for them is that they are used and applied to study cost landscapes. They have a great deal of potential, but it is not possible to judge their strengths and weaknesses until they are used to study some real questions.

Bibliography

J. E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In J. J. Grefenstette, editor, *Proc. of the 2nd Intl Conf on GA*, pages 14–21. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 1987. ISBN 0-8058-0158-8.

O. M. Becker and M. Karplus. The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics. *The Journal of Chemical Physics*, 106, 1997.

W. Benfold, A. Prügel-Bennett, and J. Hallam. Optimal Parameters for Search Using a Barrier Tree Markov Model. *Theoretical Computer Science*, 2005a. Submitted.

W. Benfold, A. Prügel-Bennett, and J. Hallam. Optimisation of Annealing Schedules for Large Problems. In *IEEE Congress on Evolutionary Computation*, pages 1119–1126. IEEE, September 2005b. ISBN 0-7803-9363-5.

B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.

C. Bridges and D. E. Goldberg. An analysis of reproduction and crossover in a binary-coding Genetic Alogorithm. In J. Grefenstette, editor, *Proc. 2nd Inernational Conference on Genetic Algorithms and Their Applications*, 1987.

S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proc. Third Ann. ACM Symp. Theor. Comput.*, pages 151–158. ACM Press, 1971.

G. Dantzig, R. Fulkerson, and S. Johnson. Solutions of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1995. Dissertation Abstracts International 36(10), 5140B; UMI 76-9381.

C. Flamm, W. Fontana, I. L. Hofacker, and P. Schuster. RNA Folding at Elementary Step Resolution. *RNA*, 6:325–338, 2000.

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

M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

P. Garstecki, T. X. Hoang, and M. Cieplak. Energy landscapes, supergraphs, and "folding funnels" in spin systems. *Physical Review E*, 60:3219, 1999.

F. Glover. Tabu search:part i. *ORSA Jounal of Computing*, 1:190–206, 1989.

F. Glover. Tabu search:part ii. *ORSA Jounal of Computing*, 2:4–32, 1990.

D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 74–88. Morgan Kaufmann Publishers, 1987.

D. E. Goldberg and P. Segrest. Finite markov chain analysis of genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 1–8. Cambridge, 1987.

J. Hallam and A. Prügel-Bennett. Barrier trees for search analysis. In E. Cantú-Paz, editor, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1586–1587, Chicago, 12-16 July 2003. Springer-Verlag. ISBN 3-540-40603-4.

J. Hallam and A. Prügel-Bennett. Barrier Based Models of Hard Problems and Crossover. In *IEEE Congress on Evolutionary Computation*, pages 1661–1666. IEEE, September 2005a. ISBN 0-7803-9363-5.

J. Hallam and A. Prügel-Bennett. Constructing Model Problems Based on Hard Optimisation Problems. *Theoretical Computer Science*, 2005b. Submitted.

J. Hallam and A. Prügel-Bennett. Large Barrier Trees for Studying Search. *IEEE Transactions on Evolutionary Computation*, 9(4):385–397, 2005c.

K. H Hoffmann and P. Salamon. The Optimal Simulated Annealing Schedule for a Simple Model. *Journal of Physics A*, 23:3511–3523, 1990.

J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press (Ann Arbor), 1975.

T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, March 1995.

Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrand, Princeton, NJ, 1960.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163):1297–1301, 27 May 1994.

T. Klotz and S. Kobe. Exact low-energy landscape and relaxation phenomena in Ising spin glasses. *Acta Physica Slovaca*, 44:325–338, 1994a.

T. Klotz and S. Kobe. "valley structures" in the phase space of a finite 3D Ising spin glass with $\pm i$ interactions. *J. Physics A*, 27:95–100, 1994b.

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

A. Land and A. Doig. An automatic method of solving discrete programming problems. *Econometrika*, 28(3):497–520, 1960.

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, 1992a.

M. Mitchell, S. Forrest, and J. H. Holland. The Royal Road for Genetic Algorithms: Fitness Landscapes and GA performance. In F. J. Varela and P. Bourgine, editors, *Proc. of the First European Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1992b. MIT Press.

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

D. S. Moore. *Goodness-of-Fit Techniques.* Marcel Dekker, New York and Basel, 1986.

A. E. Nix and M. D. Vose. Modelling genetic algorithms with markov chains. *Genetic Algorithms in Artificial Intelligence*, 5, 1992.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

A. Prügel-Bennett. Symmetry Breaking in Population Based Optimisation. *IEEE Transactions on Evolutionary Computing*, 8(1):63–79, February 2004a.

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

M. Rattray. *Modelling the Dynamics of Genetic Alogirithms using Stastical Mechanics*. PhD thesis, University of Manchester, 1996.

C. R. Reeves and A. V. Eremeev. Statistical analysis of local search landscapes. *Journal of the Operational Research Society*, 55(7):687–693, 2004.

J. E. Rowe. Population fixed-points for functions of unitation. In C. Reeves and W. Banzhaf, editors, *Foundations of Genetic Algorithms*, volume 5. Morgan Kaufmann, San Francisco, 1998.

J. E. Rowe and C. C. J. Moey. Population aggragation based on fitness. *Natural Computing*, 3(1):5–19, 2004.

W. M. Spears. A compression algorithm for probability transition matrices. *SIAM Matrix Analysis and Applications*, 20(1):60–77, 1998.

W. M. Spears and K. de Jong. Analyzing GAs using markov models with semantically ordered and lumped states. In *Foundations of Genetic Algorithms*, pages 85–100. Morgan Kaufman, San Francisco, 1996.

P. F. Stadler, W. Hordijk, and J. F. Fontanari. Phase transition and landscape statistics of the number partitioning problem. *Phys. Rev. E*, 67:056701,1–6, 2003.

P. F. Stadler and G. P. Wagner. The Algebraic Theory of Recombination Spaces. *Evolutionary Computation*, 5:241–275, 1998.

Peter F. Stadler. Fitness landscapes arising from the sequence-structure maps of biopolymers. *J. Mol. Struct. (THEOCHEM)*, 463:7–19, 1999. Santa Fe Institute Preprint 97-11-082.

M. A. Stephens. EDF statistics for goodness-of-fit and some comparisons. *Journal of the American Statistical Association*, 69:730–737, 1974.

E. van Nimwegen, J. P. Crutchfield, and M. Mitchell. Statistical Dynamics of the Royal Road Genetic Algorithm. *Theoretical Computer Science*, 229(1-2):41–102, 1999.

M. Vose and G. Liepins. Punctuated Equilibria in Genetic Search. *Complex Systems*, 5:31–44, 1991.

Michael D. Vose. *The Simple Genetic Algorithm : Foundations and Theory.* Complex Adaptative Systems. Bradford Books, 1999.

R. A. Watson. Analysis of recombinative algorithms on a non-separable building-block problem. In K. de Jong, R. Poliand, and J. E. Rowe, editors, *Foundations of Genetic Algorithms*, pages 69–89. Morgan Kaufman, San Francisco, 2001.

D. Whitley. An Executable Model of a Simple Genetic Algorithm. In D. Whitley, editor, *Foundations of Genetic Algorithms 2.* Morgan Kaufman, San Francisco, 1993.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

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

W. Zhang and R. E. Korf. A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, 81:223–239, 1996.