Evolutionary Algorithms, Fitness Landscapes and Search

by

Terry Jones

M.Math., Computer Science, University of Waterloo, 1988.

B.Sc. (Hons), Computer Science, Sydney University, 1986.

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy in Computer Science

The University of New Mexico
Albuquerque, New Mexico

May, 1995

For me.

Acknowledgments

I have somehow managed to spend six years, in three universities, working towards a Ph.D. After a false start at the University of Waterloo, I enrolled at Indiana University and subsequently transferred to the University of New Mexico after leaving to do research at the Santa Fe Institute. In the course of these wanderings, I have been fortunate to interact with many people who have influenced me greatly. One of the pleasures of finally finishing is this opportunity to thank them.

At Waterloo, Charlie Colbourn was the first professor who regarded me as a friend rather than as a chore. Charlie inspired me to work on algorithms in a rigorous way, taught me what a proof is, and treated me as though what I thought was important. He apparently effortlessly supervised ten graduate students and produced many papers while always having time to talk, eat lunch, share a joke, and even help students move house.

Soon after arriving at Waterloo, I was adopted by Gregory Rawlins, who unexpectedly invited me to live at his apartment, after noticing that I was living in the Math building, using the suspended ceiling as a wardrobe. Gregory introduced me to posets and optimality in comparison-based algorithms. More than anyone, he has influenced the research directions I have taken over the years. He subsequently became my supervisor at Indiana and remained on my dissertation committee when I transferred to UNM. Gregory taught me to ask interesting questions, showed me what quality was, and encouraged my occasionally odd approaches to problem solving.

I was also greatly influenced by Ian Munro, who supervised my masters work and my first attempt at a Ph.D. Ian's ability to rapidly assess the worth of ideas and algorithms is amazing. I would spend a week or two on an approach to a problem and Ian could understand it, deconstruct it and tell me (correctly) that it wouldn't work in about a minute. On one memorable occasion, I went into his office with an algorithm I had devised and worked on furiously for at least two weeks. Ian listened for his customary minute, leaned back in his chair, and said "Do you want me to tell you smallest n for which it will not be optimal?"

Andrew Hensel, with whom I shared so much of my two and a half years at Waterloo, was the most original and creative person I have ever known well. Together, we dismantled the world and rebuilt it on our own crazy terms. We lived life at a million miles an hour and there was nothing like it. Five years ago, Andrew killed himself. There have been few days since then that I have not thought of him and the time we spent together.

At Indiana University, I was fortunate to be supervised by Douglas Hofstadter. Doug has both wide-ranging interests and the ability to think (and write!) clearly about his interests, in a highly original and probing way. Doug was very supportive of my extracurricular activities. When I began spending six hours a day juggling and unicycling; rather than telling me to do some work, Doug became the faculty advisor for the IU juggling club. Doug is an intelligent, kind, thoughtful and generous person. Perhaps most importantly, it was through Doug that I met Ana.

When I was offered a one year position as a graduate fellow at the Santa Fe Institute, I had barely heard of the place. Assured by many that this was an opportunity that shouldn't be missed, I left Indiana. After two days at SFI I had decided that I never wanted to leave. I have been extraordinarily lucky to have spent the last three years at SFI. There are so many wonderful things that could be said about SFI—the people, the lack of hierarchy, the open doors, the never-ending conversations, the atmosphere, the lack of red tape, the many workshops, countless interesting visitors and the diversity of interests. It is hard to imagine a more stimulating and encouraging academic environment. It will be difficult and sad to leave. Mike Simmons, the academic vice-president, is one of the many extremely competent people running SFI. The ongoing health of the institute is, I believe, largely due to Mike's vision of what SFI should be, to his refusal to allow that vision to be compromised, to his warm and friendly personality, and to his ability to deal calmly and rationally with any situation. He is fortunate to be surrounded by a team of people who combine to make SFI the unique place that it is.

At SFI I began to work with John Holland and Stephanie Forrest developing John's "Echo" system. My Echo experience has taught me many things about evolutionary biology, modeling and computer science. John always encouraged me to "follow my nose." When it led me away from Echo, he continued to support my investigations and take an active interest in them. His intuitions about Echo and genetic algorithms are extremely sharp. I have lost count of the number of times John suggested correct explanations for my observations and made accurate predictions about these systems.

After working on Echo with John and Steph for a year, I transferred schools, to the University of New Mexico. Steph became my "advisor" and I began work on fitness landscapes. Steph was my seventh advisor and, fortunately, she carried a large whip. Steph is probably the only one of the seven that could have made me finish. She has a refreshingly healthy, carefree, somewhat cynical view of the world and is great to laugh and joke with. Yet when she says "Jump!" I can only answer "How high?" This combination of fun and a kick in the pants once a week was the ideal foil to my recreational tendencies and limited attention span. It was Steph who insisted that I look for a connection between my landscapes and AI search techniques. I have come to respect and trust her judgement and to think of her as a good friend, collaborator and confidant.

Of the universities I have attended, UNM's computer science department is by far the smallest and least well funded. However, it has been at UNM though that I have finally found myself surrounded by other graduate students with whom I was inclined to interact academically. The Friday meetings of the "adaptive" group have been the highlight of my UNM time and I thank all the members of the group for keeping life interesting. In particular, I'd like to thank my evil twins, Derek Smith and Ron Hightower. Together we have spent many enjoyable hours rollerblading at high speed on campus and talking about our research. Derek in particular has always been very quick to understand my ideas about fitness landscapes. He has helped me to see where I was wrong and why I was occasionally right. He read this entire work and greatly helped with its content, style, and appearance. This dissertation would have been possible without Derek, but far less likely. Derek and Ron both have broad knowledge of computer science and possess high-speed, accurate, research quality detectors. They are full of interesting ideas and suggestions, and are models of academic generosity.

It is not possible to summarize Ana Mosterín Höpping and her influence on me in one paragraph, but I will try. Ana has completely changed my life over the last three years. Her love, intelligence, honesty, goodness, healthiness, humor, taste, liveliness and beauty have given me something to live for. She is the most balanced and well-adjusted person I have ever known. The fact of her existence is a continual miracle to me. She has supported me in hundreds of ways throughout the development and writing of this dissertation.

My parents are also very special people. My mother once told me that the last thing she and I had in common was an umbilical cord. Despite this, we are very close and I love them dearly. They have given their unconditional support, knowing that doing so contributed greatly to my absence these last nine years. They were strong enough to let me go easily, to believe in me, and to let slip away all those years during which we could have been geographically closer and undoubtedly driving each other crazy.

The other members of my dissertation committee, Ed Angel, Paul Helman, George Luger and Carla Wofsy were very helpful. They were all interested in what I was doing and improved the quality of what I eventually wrote. Ed's very healthy, even robust, cynicism and his willingness to give me a hard time at every opportunity, is the kind of attitude that I appreciate and enjoy the most. Paul made very detailed comments on all aspects of the dissertation, and tried to keep me honest with respect to connections to dynamic programming and branch and bound. George refused to let me get away with murder in the research hours I took with him, and made sure that I did a good job of understanding AI search algorithms. Though Carla was a late addition to the committee, she quickly and happily read the dissertation.

Joe Culberson, of the University of Alberta, is one of the few people who appreciated and fully understood what I was trying to achieve in this dissertation. Joe and I share similar views on many aspects of fitness landscapes. I had known Joe for years before I was pleasantly surprised to discover that we had independently conceived of a crossover landscape in essentially the same way. We have exchanged many ideas about landscapes for the last two years, keeping careful track of each other's sanity. My thinking and writing are clearer as a result of Joe's rigor and insistence that I make myself clear to him.

At SFI I have profited from many discussions with Melanie Mitchell, Chris Langton, Richard Palmer, Una-May OReilly, Rajarshi Das, Michael Cohen, Aviv Bergman, Peter Stadler, Walter Fontana, Stu Kauffman, and Wim Hordijk (who also read and commented on the first two chapters of this dissertation). I have made friends and received encouragement from many people in the research community, including David Ackley, Ken De Jong, David Fogel, Jeff Horn, Rich Korf, Nils Nilsson, Nick Radcliffe, Rick Riolo, Günter Wagner and Darrell Whitley. Jesús Mosterín read my thesis proposal, cheerfully scribbled “nonsense!” across it, and then sent me a useful page of formal definitions. Many members of the staff at SFI, especially Ginger Richardson, have become good friends. The CS department at UNM has been wonderful; I have been particularly helped by Ed Angel, Joann Buehler, Jim Herbeck and Jim Hollan.

Finally, I have made many friends along the way. They have helped me, one way or another, in my struggle to complete a Ph.D. Many thanks to Sandy Amass, Marco Ariano, Marcella Austin, Amy Barley, Greg Basford, Dexter Bradshaw, Ted Bucklin, Peter Buhr, Susy Deck, Emily Dickinson, Elizabeth Dunn, Beth Filliman, Bob French, Julie Frieder, Elizabeth Gonzalez, Fritz Grobe, Steve Hayman, Ursula Höpping, Helga Keller, The King, Jim Marshall, Gary McGraw, Heather Meek, Eric Neufeld, Luke OConnor, Lisa Ragatz, Steven Ragatz, Kate Ryan, Philip San Miguel, John Sellens, Francesca Shrady, Lisa Thomas, Andre Trudel and the one and only Françoise Van Gastel.

Evolutionary Algorithms, Fitness Landscapes and Search

by

Terry Jones

B.Sc. (Hons), Computer Science, Sydney University, 1986
M.Math., Computer Science, University of Waterloo, 1988
Ph.D., Computer Science, University of New Mexico, 1995

ABSTRACT

A new model of fitness landscapes suitable for the consideration of evolutionary and other search algorithms is developed and its consequences are investigated. Answers to the questions "What is a landscape?" "Are landscapes useful?" and "What makes a landscape difficult to search?" are provided. The model makes it possible to construct landscapes for algorithms that employ multiple operators, including operators that act on or produce multiple individuals. It also incorporates operator transition probabilities. The consequences of adopting the model include a "one operator, one landscape" view of algorithms that search with multiple operators.

An investigation into crossover landscapes and hillclimbing algorithms on them illustrates the dual role played by crossover in genetic algorithms. This leads to the "headless chicken" test for the usefulness of crossover to a given genetic algorithm and to serious questions about the usefulness of maintaining a population. A "reverse hillclimbing" algorithm is presented that allows the determination of details of the basin of attraction of points on a landscape. These details can be used to directly compare members of a class of hillclimbing algorithms and to accurately predict how long a particular hillclimber will take to discover a given point.

A connection between evolutionary algorithms and the heuristic search algorithms of Artificial Intelligence and Operations Research is established. One aspect of this correspondence is investigated in detail: the relationship between fitness functions and heuristic functions. By considering how closely fitness functions approximate the ideal for heuristic functions, a measure of search difficulty is obtained. This measure, fitness distance correlation, is a remarkably reliable indicator of problem difficulty for a genetic algorithm on many problems taken from the genetic algorithms literature, even though the measure incorporates no knowledge of the operation of a genetic algorithm. This leads to one answer to the question "What makes a problem hard (or easy) for a genetic algorithm?" The answer is perfectly in keeping with what has been well known in Artificial Intelligence for over thirty years.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

LIST OF TABLES

Table Page

Table Page

LIST OF ALGORITHMS

Algorithm Page

ABBREVIATIONS

The following abbreviations are used in this dissertation:

§x(y)	Section x on page y.
β	The bit-flipping operator. §2.6.1(31)
μ	A generic mutation operator. §2.6.2(32)
ϕ	A generic operator. §2.3(24)
ψ	A generic operator. §2.3(24)
$\phi_{k \to l}$	Class of fixed cardinality operators with cardinality k and l. §2.5.9(29)
χ	A generic crossover operator.
$\chi_n^{k \to l}$	n-point crossover producing l offspring from k parents. §2.6.3(34)
$\chi_u^{k \to l}$	Uniform crossover producing l offspring from k parents. §2.6.3(34)
AI	Artificial Intelligence.
$B_\phi(v)$	The basin of attraction of a point v under an operator ϕ. §2.5.8(29)
BH	Bit-flipping Hillclimbing. §3.6(55)
CH	Crossover Hillclimbing. §3.7(57)
EC	Evolutionary Computation. §1.1(4)
EP	Evolutionary Programming. §1.1(4)
ES	Evolution Strategy. §1.1(4)
$\mathcal{F}$	Fitness space. §2.3(24)
GA	Genetic Algorithm. §1.1(4)
GP	Genetic Programming. §1.1(4)
$G_\mathcal{L}$	The graph corresponding to a landscape $\mathcal{L}$. §2.4(26)
$\mathcal{L}$	A landscape. §2.4(25)
LSA	Local search algorithm. §4.2(86)
$\mathcal{M}(S)$	The set of all multisets whose elements are drawn from a set S. §2.2.1(18)
$\mathcal{M}_q(S)$	Those elements of $\mathcal{M}(S)$ with cardinality q. §2.2.1(18)
$N_\phi(v)$	The neighborhood of a vertex v under the operator ϕ. §2.3(24)
$\mathcal{O}$	Object space. §2.3(20)
OR	Operations Research.
$>_S$	A partial order on a set S. §2.2.2(18)
P	Probability.

$\mathcal{R}$	Representation space. §2.3(20)
TM	Turing Machine.
TSP	Traveling Salesperson problem.
$u(x)$	The number of ones in a binary string x.

CHAPTER 1

Introduction

The natural world exhibits startling complexity and richness at all scales. Examples include complex social systems, immune and nervous systems, and the intricate interrelationships between species. These are but a few of the wonders that have become more apparent as we have increased our ability to examine ourselves and the world around us. Science is an ever-changing system of beliefs that attempts to account for what we observe, and it has adjusted itself dramatically to accommodate incoming data. It is remarkable that so much of what we see has come to be explained by a single theory: the theory of evolution through variation and selection.

Evolution through natural selection has profoundly affected our view of the world. The society into which Charles Darwin delivered what has become known as *The Origin of Species* in 1859 was primed for this change [1]. At that time, theories of the mutability of species were rumbling through all levels of English society. Such suggestions were in direct confrontation with the teachings of the church and were therefore an intolerable affront to the position of God in the universe. In 1844, *Vestiges of the Natural History of Creation* [2], published anonymously, had scandalized the church and academia, and fanned the flames of unrest in the lower classes. In those times, science and morality were tightly coupled. Many fields of scientific enquiry currently enjoy freedom of thought in an atmosphere that owes much to the revolution wrought by the theory of evolution. It is impossible to fully appreciate the importance and magnitude of these changes without an understanding of the social context in which this theory developed.

Darwin, like others who promoted evolution through natural selection, was by no means infallible. For example, he was unable to demonstrate a mechanism of inheritance in which variation was maintained. His eventual hypothesis on the subject, the theory of *pangenesis* proved incorrect. It was fifty years before the details of hereditary began to fall into place, and another thirty before the "evolutionary synthesis" solidified the link between the theory of evolution and the relatively young field of genetics [3, 4]. Nevertheless, Darwin had identified the central mechanism of evolution: selection in the presence of variation, or "descent with modification" as he called it. In many cases, the specifics of evolution through variation and selection are still being brought to light. However, the basic mechanisms have sufficed to explain, to the satisfaction of many, an incredibly wide range of behaviors and phenomena observed in the natural world.

It is no wonder then that computer scientists have looked to evolution for inspiration. The possibility that a computational system, endowed with simple mechanisms of variation and selection, might be persuaded to exhibit an analog of the power of evolution found in natural systems is an idea with great appeal. This hope has motivated the development of a number of computational systems modeled on the principles of natural selection. Some history of these efforts is described by Goldberg [5] and Fogel [6].

A difficulty with the hope that we might build computational systems based on the principles of natural selection, and put these systems to practical use, is that natural systems are undirected, whereas virtually all our activity is highly directed. We use the computer as a tool for solving specific problems we construct, and we place a lot of emphasis on doing so as quickly as possible and with as little waste as possible. Natural systems have no such aims or constraints. Survival is not directed toward some fixed target. Instead, it is a matter of keeping a step ahead, in any accessible direction. It is a gross generalization, but I believe efforts to date can be grouped into two broad categories:

- Useful systems that are modeled loosely on biological principles. These have been successfully used for tasks such as function optimization, can easily be described in non-biological language, and are often outperformed by algorithms whose motivation is not taken from biology.

- Systems that are biologically more realistic but which have not proved particularly useful. These are more faithful to biological systems and are less directed (or not directed at all). They exhibit complex and interesting behaviors but have yet to be put to much practical use.

Of course, in practice we cannot partition things so neatly. These categories are merely two poles, between which lie many different computational systems. Closer to the first pole are the *evolutionary algorithms*, such as Evolutionary Programming, Genetic Algorithms and Evolution Strategies. Closer to the second pole are systems that may be classified as Artificial Life approaches [7]. Both poles are interesting in their own right, and I have spent time exploring aspects of each. I have nothing against evolutionary algorithms that are used to optimize functions or against artificial life systems that are difficult to put to practical use. Yet it seems to me that we are still rather far from tapping the best of both worlds and the hope of being a part of that research, when and if it happens, is the ultimate motivation for the current work.

This dissertation is primarily concerned with evolutionary algorithms. A non-biological perspective on these algorithms is developed and several consequences of that position are investigated. It is important to remember that this is simply one perspective on these algorithms. They would not exist without the original biological motivation, and that ideal has prompted this investigation. I believe that it is important to maintain the ideal and simultaneously keep careful track of whether real progress has been made. Evolutionary algorithms are examined through the study of their relationship to two things:

1. The biological notion of a fitness landscape.

2. Other search algorithms, particularly various hillclimbers and heuristic state space search algorithms found in artificial intelligence and operations research.

The first of these relationships, the connection between evolutionary algorithms and fitness landscapes, has achieved almost folkloric recognition in the evolutionary computation community. Practically every researcher in that community is familiar with the landscape metaphor and does not express discomfort or concern when the descriptions of the workings of these algorithms employ it extensively, using words such as "peaks," "valleys" and "ridges." Surprisingly however, it is rare, within the field of evolutionary algorithms, to find an actual definition of a fitness landscape. This situation has perhaps developed because everyone grasps the imagery immediately, and the questions that would be asked of a less evocative term are not asked. This dissertation presents a model of landscapes that is general enough to encompass much of what computer scientists would call search, though the model is not restricted to either the field or the viewpoint. Evolutionary algorithms are examined from the perspective the model affords.

The second relationship that is addressed is the connection between evolutionary algorithms and the search algorithms of artificial intelligence and operations research. This relationship is not widely recognized. I believe it is a an important one. Its identification allows discussion of evolutionary algorithms in non-biological terms. This leads to a view of evolutionary algorithms as heuristic state space search algorithms, and this correspondence is used to throw light on the question of what makes a problem hard (or easy) for a genetic algorithm.

The remainder of this chapter provides a brief introduction to evolutionary algorithms and fitness landscapes, the model of computation, a discussion of the effects of choice of language, an argument for simplicity, a brief outline of related work, and a description of the structure of the dissertation.

1.1. Evolutionary Algorithms

There is no strict definition of the class of algorithms known as "evolutionary algorithms." It is not hard to argue that members of the class have little to do with biotic evolution and are not strictly algorithms. Nevertheless, the term exists, and I will attempt to make clear what is usually meant by it. In rough order of appearance, members of the class generally fall into one of the following three categories: Evolutionary Programming [8], Genetic Algorithms [9] and Evolution Strategies [10, 11]. The collective set of approaches has become known as Evolutionary Computation (see [6, 12] for overviews).

Many variants of these methods have emerged. Two of the best known are Messy Genetic Algorithms [13] and Genetic Programming [14, 15], both of which are descendants of genetic algorithms. In describing evolutionary computation, some would fling the net even more widely, and include neural networks, simulated annealing, and many models from Artificial Life. I will not consider these extensions in the following description, but many of the results of this dissertation bear on those fields, and the connections will be detailed in later chapters. The following description of an algorithm could be applied to any of the above evolutionary algorithms or their variants:

> The algorithm maintains a collection of potential solutions to a problem. Some of these possible solutions are used to create new potential solutions through the use of *operators*. Operators act on and produce collections of potential solutions. The potential solutions that an operator acts on are selected on the basis of their quality as solutions to the problem at hand. The algorithm uses this process repeatedly to generate new collections of potential solutions until some stopping criterion is met.

The above description is couched in a deliberately neutral language. In practice, these algorithms are often described in biological terms. The algorithms maintain not a collection of potential solutions, but a *population* of *individuals*. The problem to be solved is given by a *fitness function*. Operators are *genetic operators* such as *mutation*, *recombination* (or *crossover*) and *inversion*. Individuals are chosen to *reproduce* on the basis of *selection* based on their *fitness*. In each case, these algorithms were designed to mimic aspects of evolution through natural selection with the hope that this would lead to general methods that would prove widely useful through similar operation. Although the algorithms represent vast simplifications of the actual evolutionary processes, they have proved widely successful.

1.2. Fitness Landscapes

Using a landscape metaphor to develop insight about the workings of a complex process originates with the work of Sewall Wright in the mid–1920s [16]. By the time Wright first published on the subject, Haldane [17] had begun to think along similar lines. Wright used landscapes to provide easily accessible imagery for his theory of speciation. Since that time, biologists have frequently used the metaphor. Landscapes, according to Eldredge [18] are "By all odds the most important metaphor in macroevolutionary theory of the past fifty years." There are two main interpretations of landscapes in biology. In the first, each member of a population is considered a point on a fitness landscape. This imagery has been used extensively in the development of evolutionary thought. It is the foundation of both Wright's *Shifting Balance* and Simpson's *Quantum Evolution* models of speciation [16, 19]. In the second approach, a point on the landscape is taken to represent the average fitness of an entire population which moves on a surface of mean fitness. Population geneticists have used this viewpoint to prove theories about the effects of selection on average fitness [20, 21].

To biologists, going uphill on a landscape is thought of as increasing fitness. Physicists also find the landscape metaphor useful, but they are usually interested in low energy states of some material (e.g., a spin glass [22, 23]), and so consider lower points on the landscape more desirable. Computer scientists are divided. The conflict is of course a trivial one and it vanishes if one talks exclusively of optimizing on a landscape. I will defer to biology. In this dissertation, algorithms are seeking to find high points on the landscape. Formal discussion of landscapes will begin in Chapter 2.

1.3. Model of Computation

Establishing a formal model of computation is difficult and somewhat arbitrary at the best of times. Even when the computation is being performed in a well-defined manner on a well-defined computational device, there are many subtle issues whose resolution, one way or another, may have decidedly non-subtle effects [24]. It is no surprise then that evolutionary algorithms have not been the subject of wide attention when it comes to models of computation.

When it is discussed, the model of computation that is often used when discussing evolutionary algorithms is some variant of a *black-box* model. See, for example, [5, 25, 26, 27, 28, 29]. In this model, one is presented with a black box whose properties are unknown. The box accepts inputs and each input is rewarded with the value of some mysterious function. The aim of this series of probings is also given, and might be "Find an input which produces a large function value" or possibly "Find an input that produces an output value greater than 100."

Notice that the possible vagueness of the aim of this enterprise may prevent the provider of inputs from knowing when, if ever, it has located an input (or set of inputs) that solve the search problem. To be more specific, I will call the input provider the "searcher" and the task to be addressed the "problem." This is in keeping with the subject matter of the rest of this dissertation: Search algorithms and problems. The black-box model is more commonly phrased in terms of "functions" and "optimization."

Given such a model, the operation of providing an input and receiving an output, often called performing a *function evaluation*, becomes the coin of the realm. Algorithms may be compared on this basis—for example by comparing the number of evaluations required, on average, to achieve a certain performance level, or by computing statistics based on the function values achieved over time [30]. In this dissertation, function evaluations, or, more simply, evaluations, will form the basis for the comparison of algorithm performance. I will adopt a form of black-box model, but before doing so, I wish to argue that without thought, it can be misleading in two important ways:

1. **Representation.** The first problem with the abstract black-box model of computation arises from the fact that we rarely encounter black boxes in the real world. The model assumes that the problem to be solved is already embodied in some mechanical device and that our only task is to play with the input knobs of the box. In reality, we encounter problems in the world outside the computer, and if we intend to attempt to solve them using a computer, we have to get them into the computer. The black-box model is silent on this aspect of computational problem-solving even though this step is an inevitable and important part of the problem solving process. The need to formulate the problem in a fashion that is suitable for computation requires making a decision about how to represent the input to the black box. This is a non-trivial problem. But

the black-box model obscures this requirement, by providing us with a physical object with a conveniently predefined interface. This choice needs to be made every time we address a search problem computationally. The problem of representation has been recognized and debated in the field of evolutionary computation. I am trying to make two points: that the black-box model does not draw attention to this choice and that the choice must always be made.

The choice that is made may have a dramatic impact on how easily the problem will be solved. This adds a very human factor to our otherwise impersonal black-box model. When faced with a search problem such as "Find two integers whose product is 15 and whose sum is 8," few people, presumably, will rush off and construct an elaborate encoding of a pair of integers and write a genetic algorithm to search for the solution. When asked to "Find an arrangement of eight queens on a chessboard such that no queen is attacking any other" it is not unreasonable to use as a representation an eight by eight array of computer words, exactly eight of which are non-zero. Others would immediately see that possible solutions could be represented via a permutation of the integers one to eight, which leads to a far more elegant search algorithm [31]. The same black box could be used to evaluate board configurations for feasibility, but the difference in the choice of representation makes the second approach far more promising than the other. John Von Neumann would perhaps not have seen any need for representation to solve this problem. An extensive discussion of representation and its effects on the difficulty of the "Missionaries and Cannibals" problem is given by Amarel in [32]. Algorithms that operate within the black-box model are often said to be performing *blind* search [5, 27, 28]. The above discussion illustrates that care should be taken with this term since the degree to which the algorithm is fumbling in the dark will very much depend on how much the designer of the algorithm could see.

2. **Evaluation.** The second potential problem with the black-box model concerns the output of the box and is of a similar nature. The severity of this problem depends to some extent on the nature of the original problem statement. In many search problems, no function is given, and again one must make a choice. A good example is the optimization formulation of the NP-Hard Traveling Salesperson Problem (TSP): "Given n cities and their inter-city distances, find a Hamiltonian circuit whose total length is minimal." In this case, we are not told how to recognize a solution and we are certainly *not* given a function that we can interrogate in the black-box fashion.

In some cases, such as in TSP, a seemingly appropriate function will be so obvious that it will be easy to believe that it was always there. It was not—look at the problem statement. In evolutionary algorithms, TSP has been, at least to my knowledge, universally approached as though a black box was given that took a Hamiltonian circuit as input and returned the total length of that circuit. Again I want to emphasize very strongly that this is a choice (it is often a very good choice). To do this, consider that there is a vastly superior choice: A function which returns the exact number of city exchanges that are required to transform the input into the optimum. Why then do we not use this black box instead of the path length version? After all, the problem could be solved in polynomial time with the simplest form of hillclimbing given such a black box. The reason, of course, is that we cannot *build* such a black box. It should be apparent that the black box is constructed by the problem solver and that we have a choice in how that is done.

A good example of a problem in which we are not given any function is Rubik's Cube. Though this is not a problem (to the best of my knowledge) that has been addressed using evolutionary algorithms, it has received wide attention in the artificial intelligence community. No-one has found a heuristic function whose value can be used successfully by any known search algorithm. Korf [33] presents four heuristic functions for the simplified $2 \times 2 \times 2$ cube, none of which has been used successfully to guide a heuristic search on this simpler problem. To solve Rubik's cube using a non-exhaustive search on a computer, we have to make a choice about how we will assess the worth of the cube configurations we encounter. The reason the puzzle is so hard is that it is not clear how to do this. A similar difficulty was identified for 26 transistor placement heuristics by Shiple et al. [34].

Putting these points together, to use an algorithm that operates via interrogating a black box, we find that we must first construct the black box itself. To make an unqualified claim that an algorithm is worthwhile because of its general-purpose nature and that it can be used in a black-box situation is to ignore two important aspects of problem solving. Instead of stubbing our toe on a black box while strolling casually through a field armed only with a blind algorithm, we build a black box, half-bury it in a field and then, as if by chance, stumble across it and whip out our general-purpose, robust, blind search algorithm and use it to extract the secrets of the black box, after which we proclaim the great and general usefulness of our weapon.

In summary, both these arguments claim that something the black-box model appears to provide actually has to be constructed by the problem solver. These are the form of the input, and the output function. In other words, the entire black box. A naive consideration of the black-box model would take those two as given, whereas I believe that both of them must always be constructed. This leaves us in a curious position. The model of computation we had intended to use has seemingly vanished. There is no black box until the problem solver creates it. The black box model is not independent of the problem to be solved in the way the comparison-based model of traditional analysis of algorithms is [24]. The comparison based model is useful for this reason. It is only fair to compare algorithms based on the number of evaluations they perform if they are using the same input representation and evaluation function.

A graphic illustration of the impact of these choices is provided by comparing exhaustive search on the alternate representations of the eight queens problem mentioned above. One representation has a search space of size 4,426,165,368 while the other is of size 40,320. Even comparing the same algorithm under a different representation leads to a difference in expected performance of five orders of magnitude. If we were to extend such comparisons to different algorithms employing different representations and fitness functions, our results could be even more wildly misleading.

Our model of computation then shall be based on evaluations, and we can use it to compare algorithms, provided that the choices of representation and evaluation function have been made and are the same for the algorithms in question. We will acknowledge the very important human component present in the construction of a black box. The process of choosing representation and choosing an evaluation function requires inferencing and insight into the problem, and may be more important than the eventual choice of algorithm to interpret the black box. This encourages caution when making claims about the virtues of general-purpose, weak methods. These claims tend to ignore the fact that much of problem solving lies in choices that are not made by these weak methods.

1.4. Simplicity

Evolutionary algorithms, though often simple to describe and implement, have proved difficult to understand. Analysis of algorithms (a phrase which, incidentally, has many wonderful anagrams) typically involves the choice of a model of computation, proofs of upper bounds on worst and possibly average-case behavior and proofs of lower bounds on problem difficulty. With luck, better lower bounds will be proved and better algorithms designed until these bounds meet, to within a constant factor, and the problem is considered solved. There are many examples of this process and many of the most basic algorithms in computer science have interesting histories.

The analysis of an evolutionary algorithm is a quite different matter. To begin with, as mentioned above, it is not even clear that they should be regarded as algorithms in the strict sense. Perhaps the biggest reason why analysis cannot proceed as above is that although the algorithm is given in advance, the details of the problem to be solved are not. In the traditional analysis of algorithms, the algorithm being analyzed is an algorithm for a specific problem. In many cases the input to the algorithm is not prespecified, but it is still almost always possible to perform worst-case analysis of the algorithm's behavior. However, worst-case analysis of algorithms that make choices based on pseudo-random numbers is typically not terribly enlightening, and this is certainly also the case with evolutionary algorithms. By considering all possible inputs to an algorithm, average-case behavior can sometimes also be determined (though this is typically a far more demanding task). The same analysis cannot be done for an evolutionary algorithm, because the statement of the algorithm gives no hint as to the identity of a specific problem that is to be solved. This does not preclude analysis completely, it just makes it more difficult and often quite unlike that seen in more traditional analysis.

A second major difficulty that confronts the would-be analyst is the degree to which the evolutionary algorithms are stochastic. In traditional analysis of algorithms, the algorithm always behaves in almost exactly the same way given the same input.[1] In evolutionary algorithms, if this *ever* happened (on a problem of even modest size), it would be considered so improbable that the implementation would immediately be checked for bugs. This massive degree of stochasticity does not lead to algorithms that are straightforward to understand or predict. Intuition is often not a particularly reliable guide to the behavior of a system whose outcome is the result of the composition of thousands or millions of small-scale chance events. A fine illustration of these difficulties is given in work on the so-called Royal Road functions by Mitchell et al. in which a seemingly simple investigation with a predictable outcome produced completely unexpected results [36].

That example had a great effect on me, and marked the beginning of what has become a very pronounced preference towards asking simple questions about these complex algorithms. Simple questions have a reasonable probability of being answerable and have at least the potential for having simple answers. If we ask a simple question of a system and receive a completely unexpected result, it seems

[1] Quicksort is a well-known example with varying behavior, but the amount of stochasticity in that algorithm is not great enough to make detailed analysis impossible [35].

logical that we should seriously question our understanding of the system, and that we should continue to ask simple questions.

This dissertation is a collection of very simple questions and my attempts to answer them. It may appear to some that these simple questions, and the methods that are used to address them, have little to do with evolutionary algorithms. I would argue strongly that this is not the case, and that these questions and the methods used to address them have *everything* to do with the more complicated algorithms. For example, it is important to study the operators of these algorithms in isolation and it is important to study the effect of a population-based search by comparing it with algorithms that have no population.

Amusingly, my taste for asking simple questions is the result of time spent in an environment in which the most complex problems are being addressed. It is not a contradiction to attempt the study of entire complex systems by a form of reductionism that proceeds by asking a lot of simple questions. I regard the development of this taste for simple questions to be (finally) a sign of increased maturity. It is the most important lesson that I will take away from three wonderful years at the Santa Fe Institute.

1.5. Related Work

A discussion of related work would make little sense at this point, and could not be related to the model that will be developed. Related work will be discussed in detail in Chapter 6. That chapter describes how this work relates to (1) Other models of landscapes that have been used in biology, chemistry, computer science and physics, and (2) Other search algorithms, particularly heuristic state space search. A full treatment of these relationships is not possible until the model of this dissertation has been presented and some of its uses have been demonstrated. The body of the dissertation contains many references to and hints at these relationships, though discussion is usually deferred.

1.6. Dissertation Outline

The following chapters can be roughly divided into three sections: formalism (Chapter 2), applications (Chapters 3 and 4 and the second half of Chapter 5) and connection with other search techniques (Chapter 5). Chapter 2 presents the landscape model, and discusses its consequences, advantages, limitations and relevance. Chapter 3 develops the *crossover hillclimbing* algorithm and examines the role of crossover, macromutations and populations in genetic algorithms. Chapter 4 demonstrates one approach to quantifying the behavior of algorithms on landscapes through the use of a *reverse hillclimbing* algorithm that can be used to compute the exact probabilities that a number of hillclimbing algorithms will reach a point on a landscape. Chapter 5 looks at the connection between evolutionary algorithms and heuristic search, and shows the extent to which evolutionary algorithms can be regarded as heuristic state space search algorithms. One aspect of the correspondence is treated in detail and results in a new answer to the question "what makes a problem hard (or easy) for a genetic algorithm?" Chapter 6 examines something of the history of the use of the landscape metaphor in biology and evolutionary algorithms, other related work on landscapes and heuristic search. Finally, Chapter 7 presents a summary of the results of this dissertation.

1.7. Abbreviations

The remainder of this dissertation uses many abbreviations for common phrases and algorithm names. I hope that these will not prove too intimidating to those not already familiar with the various evolutionary algorithms. A complete list of the abbreviations begins on page xxiv.

CHAPTER 2

A Model of Landscapes

This chapter presents the landscape model that will be used as a framework for the remainder of the dissertation. It is assumed that one's aim is to address a search problem. It is possible to adopt more neutral language, to the point that the model seems completely abstract. However, adopting the language of search does not restrict applicability too severely. One argument for neutrality is that since the model is applicable to situations that some people would not consider search, it should be described in a general way that will permit its easy application to these fields. A counter argument holds that most of what we do, especially with computers, can be phrased in terms of search. If the landscape model presented here is useful in some field of endeavor, then it is probably possible to describe that endeavor as a search. As will be seen, choosing to view a wide range of situations from a search perspective is not without precedent.

2.1. Motivation

Given the aim of this chapter, to present a model of landscapes, there are many preliminary questions that should be addressed before the model is presented. Three of these are particularly important.

- **What is a landscape?**
  A landscape is one way to view some aspects of a complex process. As such, it is a tool that we hope to use to increase understanding of the process, to suggest explanations and ideas, and to provide an intuitive feeling for those aspects of the process. This description is vague because landscapes may be used in a wide variety of situations. Wright employed the metaphor to provide a simplified and intuitive picture of his work on the mathematics of gene flow in a population [37]. We will use landscapes to produce a simplified view of some

aspects of search. The metaphor has been used in this way for some time to describe evolutionary algorithms, though rather informally. As a result, I think the current use of the metaphor does not always produce beneficial results. Support for this opinion will be presented shortly.

It is important to understand what I believe a landscape is and how that will affect the presentation of the model in this chapter. In particular, a landscape is neither a search algorithm nor a search problem. As a result, to formally describe a landscape, it is not necessary to formally describe all aspects of all search algorithms or all aspects of all search problems. For this reason, we will examine search algorithms and problems of a certain kinds and will omit many details of algorithms and problems. The landscape model will be widely applicable to search, but it is far from a complete picture of search.

A landscape is a structure that results from some of the choices (by no means all) that are made when we use a computer to search. It presents only a simplified view of aspects of the search process. A landscape is a tool and it is a convenience. The model of landscapes presented in this chapter aims to formalize the notion of a landscape and to make the result as general and useful as possible for thinking about computational search.

- **Why the interest in landscapes?**
  My interest in the landscape metaphor stems from an interest in evolutionary algorithms. In the community of researchers working on these algorithms, the word "landscape" is frequently encountered—both informally, in conversations and presentations, and formally, in conference and journal papers. It is natural therefore that one should seek to understand what is meant by the word. Remarkably, it proved difficult to find out exactly what people meant by the word, and when that could be discovered, the definitions did not concur. In my opinion, the landscape metaphor is a powerful one and because its meaning is appears intuitively clear, people are inclined to employ it rather casually. Questions that would be asked of a less familiar and less evocative word simply do not get asked. After all, everyone knows what a landscape is. Or do they? If the term was only used in an informal sense or for the purposes of rough illustration, the situation would be more tolerable. Instead, properties of fitness landscapes are used as the basis for explanations of algorithm behavior and this is often done in a careless fashion that is detrimental to our understanding. Thus, the motivation for trying to establish just what is (or should be) meant by a landscape comes from (1) the fact that the word is in widespread use, (2)

the belief that it is usually not well-defined, and (3) the belief that this lack of definition has important consequences. I will take the first of these as given and argue below that the two beliefs are justified.

- **What is wrong with the current definitions of landscapes?**
  The following are some of the current problems with the use of the landscape metaphor *in the field of evolutionary computation.* The use of landscapes in other fields will be discussed in Chapter 6. Some of the terminology used, (e.g., *operator*), will not be defined until later in the chapter.

  1. In many cases, landscapes are not defined at all. This is very common informally, but also happens quite frequently in formal settings. It is also common for papers to reference an earlier paper that contains no definition or a poor one. This practice has decreased in the last year.
  2. Another common problem is vague definitions of the word. For example, one encounters statements such as "the combination of a search space S and a fitness function $f : S \to \mathbb{R}$ is a fitness landscape." Apart from the fact that what has been defined here is simply a function, it might not be immediately clear what is wrong with such a definition. The problem is that virtually every term we would like to use in describing a landscape depends on being able to define neighborhood but the above definition tells us nothing about neighborhood. For example, perhaps the most frequently used landscape-related term is "peak." But what is a peak? A simple definition is: A point whose neighbors are all less high than it is. It is not possible to sensibly define the word "peak" without defining what constitutes neighborhood. The need to be specific about neighborhood is not widely attended to. As a result, in many cases, it is not clear what is meant by the word landscape. The expressions "fitness function" and "fitness landscape" are often used interchangeably, both formally and informally. A fitness *function* is a function: a mapping from one set of objects to another. There is no notion of neighborhood, and one is not needed. A fitness *landscape* requires a notion of neighborhood.
  3. Even when neighborhood is defined, it is often done so incorrectly. Informally, people will staunchly defend the binary hypercube as *the* landscape when an algorithm processes bit strings, even if the algorithm never employs an operator that always flips exactly one bit chosen uniformly at random in an individual. Certainly this hypercube may have vertices that fit the above definition of a peak, but the relevance of these points to such

an algorithm is far from clear. Even when mutation is employed in a GA, the mutation operator does *not* induce a hypercube graph. The mutation operator used in a GA does not always alter a single bit in an individual and hence the mutation landscape graph is not a hypercube. For binary strings of length n, the graph is the complete graph K_{2^n} augmented with a loop from each vertex to itself. With a per-bit mutation probability of p, the bidirectional edge between two vertices, v_1 and v_2 with Hamming distance d has a transition probability of $p^d(1-p)^{n-d}$. This is the probability that a single application of the mutation operator transforms v_1 into v_2 (or vice-versa). Figure 3 on page 33 illustrates such a graph.

4. Even though neighborhood is sometimes specified, and specified correctly, a single landscape is very often used as the framework for considering the operation of an algorithm that uses multiple operators. The best example is the use of a hypercube graph landscape as a basis for consideration of both mutation and crossover in a GA. Apart from the fact that the hypercube is not induced by either operator, this position leads to statements such as "crossover is making large jumps on the landscape." Crossover is viewed here in terms of the landscape structure that mutation is popularly thought to create. Mutation takes single steps, but crossover causes large jumps. Why is this? Even when the mutation probability in a GA is set to zero, people will insist on talking about local maxima from the point of view of mutation, despite the fact that these clearly have no relevance whatsoever to the algorithm. The model presented in this chapter claims that each operator is taking single steps on its own landscape graph. It is possible to consider the effect of crossover in terms of the mutation graph, but it is equally possible to consider mutation in terms of the crossover graph. The model insists that we explicitly acknowledge this, rather than, as is now common, comparing one operator in terms of the structure defined by another.

5. In the study of GAs, there is a tremendous bias towards thinking purely in terms of a mutation landscape. This bias can be difficult to detect in oneself. A good illustration comes again from the consideration of the word "peak." As discussed above, for something to be a peak, it must have neighbors. If we define neighborhood in terms of the operators we employ, it should seem strange that we can use the word peak for one operator but not for another. Why is there no notion of a peak under crossover? Why is the term reserved solely for mutation? A GA creates new individuals

through crossover and through mutation. In some instances, the number of individuals created via crossover will be greater than the number created through mutation. In these cases, shouldn't we be more concerned about peaks (local optima) with respect to crossover than with respect to mutation? In fact, if the domain of a fitness function has a point whose fitness is higher than that of all others, the mutation landscape will always have a single global optimum and no local optima, as any point can be reached from any other via one application of the mutation operator. By this reasoning, we should not be concerned with local optima from mutation's point of view, since they do not exist! The current notions of landscape pay no heed to operator transition probabilities, and this is clearly important. The model of landscapes presented in this chapter incorporates these probabilities and removes the mutation landscape bias, giving each operator its own landscape, complete with peaks.

6. The current models of landscape cannot be simply extended to consider other search algorithms. It is not even possible to use them consistently across algorithms within evolutionary computation. Current models typically require fixed dimensionality and a distance metric. This makes it difficult to describe, for instance, the landscape that genetic programming (GP) is operating on. The proposed model makes it simple to view GP as occurring on landscapes, as well as making connections to many search algorithms in other fields, such as those in Artificial Intelligence [38, 39].

2.2. Preliminary Definitions

This section defines multisets and gives a very brief introduction to relations, orders and the theory of directed graphs.

2.2.1. Multisets

A multiset is

> "...a mathematical entity that is like a set, but it is allowed to contain repeated elements; an object may be an element of a multiset several times, and its multiplicity of occurrences is relevant."
>
> Knuth [40, page 454].

For example, $\{1, 1, 4, 5, 5\}$ and $\{4, 1, 5, 5, 1\}$ are equivalent multisets. In contrast, $\{1, 1, 2\}$ and $\{1, 2\}$ are equivalent as sets, but not as multisets. Of course, any set is also a multiset. Binary operations $\cup$, $\uplus$ and $\cap$ between multisets that obey commutative, associative, distributive and other laws can be simply defined [40, page 636], though we will have no use for them in this dissertation. Knuth mentions a number of situations in which multisets arise naturally in mathematics. Kanerva makes extensive use of multisets in his analysis of Sparse Distributed Memory [41]. Given a set S, I will use $\mathcal{M}(S)$ to denote the infinite set of all multisets whose elements are drawn from S. Thus the multiset above is an element of $\mathcal{M}(\{1{,}4{,}5\})$, as are the multisets $\{1\}$, $\{4,4,4,4\}$, $\{4,4\}$ and $\{1,4,5\}$ itself. I will use $|S|$ to represent the number of elements in the multiset S. Hence, $|\{1,5,5\}| = 3$. Finally, define

$$\mathcal{M}_q(S) = \{s \in \mathcal{M}(S) : |s| = q\},$$

as the set of multisets of S with cardinality q.

2.2.2. Relations, Orders and Digraphs

A *relation*, R, between two sets A and B is a subset of $A \times B$. That is, $R \subseteq \{(a,b) \mid a \in A \text{ and } b \in B\}$. The notation aRb is used to indicate that $(a,b) \in R$. When $A = B$, R is said to be *reflexive* if $aRa\ \forall a \in A$; *symmetric* if $aRb \Rightarrow bRa\ \forall a,b \in A$; *transitive* if $aRb, bRc \Rightarrow aRc\ \forall a,b,c \in A$ and *complete* if, given $a \neq b$, at least one of aRb or bRa is true. A relation that is not complete is said to be *partial.* Different combinations of the above four properties impose different "orders" on A. If R is irreflexive, asymmetric, transitive and complete, it is called a *total order* or *complete order.* If it is irreflexive, asymmetric and transitive, but not complete, R is a *partial order.* The relation "greater than" is a total order on the set of real numbers. The relation "is an ancestor of" is a partial order on the set of all people. In this dissertation we will commonly be concerned with partial orders that are used to rank potential solutions to search problems. A relation that is reflexive, symmetric and transitive is called an *equivalence* relation. An equivalence relation creates *equivalence classes*: If R is an equivalence relation and aRb then a and b are members of the same equivalence class.

A relation R on a set A can also be thought of as a directed graph or *digraph.* The vertices of the digraph are the elements of A and there is a directed edge from vertex a to vertex b if aRb. It is traditional to exclude the possibility of edges from a vertex to itself in a digraph. This is mainly a matter of convention, and we shall allow such edges or *loops.* If R is both symmetric and complete, the digraph contains all possible edges and is denoted K_n when the digraph has n vertices. A symmetric digraph can also be regarded as an undirected *graph.* In this case, the two directed edges between a pair of vertices can be drawn as a single undirected edge. In this dissertation, we shall simply use the word "graph" to mean a directed graph. We will commonly discuss digraphs that correspond to symmetric relations and the distinction will be unimportant.

Finally, a pair of vertices v and w are *connected* if there is an undirected path between them. That is, w can be reached by traversing edges (irrespective of the directions on the edges) from v. If v is connected to w, w is therefore connected to v. The difficulty with edge directions disappears if the underlying relation is symmetric.[1] If every vertex is considered connected to itself, the relation "is connected to" is an equivalence relation on the set of vertices and thus induces equivalence classes. Each of these equivalence classes, together with the edges incident on the vertices in the class, defines a *connected component* of the graph. If a graph has a single connected component, it will be said to be *fully connected* or simply *connected.* Otherwise, the digraph is a *forest* of connected components and is said to be *disconnected.* For a much fuller discussion of connectedness in digraphs, see Harary, Norman and Cartwright [42], which also provides a good introduction to the discussion of graphs in this dissertation.

2.3. Search Problems and Search Algorithms

As mentioned in the introduction to this chapter, it will be assumed that we are interested in solving a search problem. This position has been adopted by others and, given a general interpretation of search, is not terribly restrictive. The following quotes lend support to this claim:

> "In its broadest sense, problem solving encompasses *all* of computer science because any computational task can be regarded as a problem to be solved." Nilsson [43, page 2].

> "Every problem-solving activity can be regarded as the task of finding or constructing an object with given characteristics."
> Pearl [31, page 15].

If we accept these statements, and consider the problem of "finding or constructing an object" a search problem, then we should not be surprised if very many situations

[1] This is a simplistic definition of connectedness in a digraph. However, it is sufficient for our purposes. A better treatment of this issue can be found in Harary, Norman and Cartwright [42].

can be thought of as posing search problems. As a final link in this chain of reasoning, Rich claims that

> "Every search process can be viewed as a traversal of a directed graph in which each node represents a problem state and each arc represents a relationship between the states represented by the nodes it connects."
>
> Rich [44, page 55].

If we take these three excerpts seriously, search is not only ubiquitous, but it can be described in terms of a process on a graph structure. On setting out to develop a model of fitness landscapes, I soon arrived at the conclusion that a landscape could be conveniently viewed as a graph. Some time later, it became apparent that this was a common view of search in other fields and I took pains to ensure that the model was general enough to be appropriate for discussions of these algorithms also. Given such a broad target, it is necessary to develop the model in a very general way. I am particularly concerned that this be done in a mathematically precise way that will allow formal definitions and discussion when that is needed. This is similar to the aims expressed by Banerji [45].

In setting out to present a model of computational search, there are several fundamental issues that need to be initially addressed. I assume, in line with the quotation due to Pearl above, that a search problem takes the form "Find an object with the following properties...." Then search is conducted amongst a (possibly infinite) set of objects (potential solutions). As was emphasized in §1.3(5), the first step in solving a search problem involves making a choice about the contents of this set. I will call this set of objects the *object space*, and it will be denoted by $\mathcal{O}$. This choice will often be a simple one and in many cases a good candidate for $\mathcal{O}$ will be provided in the problem statement itself. This decision is typically made before a computer is introduced as an aid to search. Example object spaces include the set of n-tuples of real numbers, the set of binary strings of length n, the set of LISP S-expressions, the set of permutations of the integers one to ten, the set of legal chess positions, the set of spin configurations in a spin glass and the set of RNA sequences.

A second step that is common when searching is to decide on some representation of the objects in $\mathcal{O}$. The representation determines a set, $\mathcal{R}$, the *representation space*, that can be manipulated more conveniently than $\mathcal{O}$ by the searcher. The model of search of this chapter does not require that $\mathcal{R} \neq \mathcal{O}$, but this will be true throughout this dissertation. In situations where $\mathcal{R} = \mathcal{O}$, (i.e., no representation is employed), one can substitute $\mathcal{O}$ for $\mathcal{R}$ in the model. When performing search via a computer, $\mathcal{R}$

will very commonly be the result of a choice of data structure, $\mathcal{D}$.[†] In this dissertation, this will always be the case. The possible ways of assigning values to the bits that comprise the memory corresponding to an instance of $\mathcal{D}$ determines $\mathcal{R}$. Elements of $\mathcal{R}$ will be used to represent elements of $\mathcal{O}$. Representation via instances of a data structure is not the only possible choice in computational search, it is merely the most common. Forms of search involving analog computation may have representations involving resistors (for example), in which case the representation space may have nothing to do with data structures [46].

Unlike the object space, the representation space is necessarily finite. Computers have finite limits and therefore, in general, it is not possible to represent the whole of $\mathcal{O}$ with $\mathcal{R}$, at any given point in time. Such a situation is very common in evolutionary algorithms. For this reason, only a finite subset $\mathcal{O}' \subseteq \mathcal{O}$ will be represented at any given time. The mapping between elements of $\mathcal{O}'$ and $\mathcal{R}$ will be called the *representation.*[2] The representation is the link between the objects we choose to examine as potential solutions to the search problem and the objects we choose to manipulate to effect the search. When the search is finished, we invert this mapping to produce the solution.

A relation, Γ, ("is represented by") between the sets $\mathcal{O}$ and $\mathcal{R}$ implements this representation. For $o \in \mathcal{O}$ and $r \in \mathcal{R}$, we will write $o\,\Gamma\,r$ to indicate that o is represented by r. The inverse relation Γ^{-1} ("represents") between the sets $\mathcal{R}$ and $\mathcal{O}$ is defined as $\Gamma^{-1} \ = \{(r,o) \mid (o,r) \in \Gamma\}$. We will write $r\,\Gamma^{-1}\,o$ to indicate that r represents o. The (possibly empty) subset of $\mathcal{R}$ representing $o \in \mathcal{O}$ will be denoted by $\Gamma(o)$ and the (possibly empty) subset of $\mathcal{O}$ that is represented by $r \in \mathcal{R}$ will be denoted by $\Gamma^{-1}(r)$. If $\Gamma(o) \neq \emptyset$, we will say that o is *represented.* If $\Gamma^{-1}(r) = \emptyset$ we will say that r is *illegal.* $\mathcal{R}' = \{r \in \mathcal{R} \mid \Gamma^{-1}(r) \neq \emptyset\}$ will be used to denote the set of elements of $\mathcal{R}$ that are not illegal. Consideration of how an algorithm restricts its attention to $\mathcal{R}'$ or manages excursions into $\mathcal{R} - \mathcal{R}'$ are discussed in §2.12(41). There is nothing to stop an algorithm from adopting a new $\mathcal{O}$, $\mathcal{O}'$, $\mathcal{R}$, $\mathcal{R}'$, or Γ at any point, for example see [48, 49, 50, 51].

At this point, we have almost enough to talk about search. We need to be slightly more specific about the nature of the search problems in which we are interested. As mentioned above, we do not need to be too specific about the details of the search problems if those details do not influence the landscapes that we will construct. In this dissertation I will be concerned with two types of search problems, which I call

[†] This does not restrict us to data structures of fixed size.

[2] Liepins and Vose have termed this the *embedding* [47].

simply *Type 1* and *Type 2*:

1. A *Type 1* search problem requires the location of an object (or objects) possessing a set P of properties. The search is only satisfied when such an object is located. The statement of the search problem includes no notion of "closeness" to a solution. Either a solution has been found and the search is successful, or one has not been found and it is unsuccessful. It is convenient to imagine the statement of the search problem as providing a function $g : \mathcal{O} \rightarrow \{0, 1\}$ defined as follows:
$$g(o) = \begin{cases} 1 & \text{if } o \text{ possesses all properties in } P. \\ 0 & \text{otherwise.} \end{cases}$$
which can be used to determine if an object satisfies the requirements of the search problem.

2. In a *Type 2* search problem, the searcher is required to find as good an object as possible (presumably within some limits on time and computation). Once again, it is convenient to imagine that the statement of the search problem provides a function $g : \mathcal{O} \rightarrow \mathcal{G}$, for some set $\mathcal{G}$, and a partial order $>_{\mathcal{G}}$ on $\mathcal{G}$ such that for $o_1, o_2 \in \mathcal{O}$, if $g(o_1) >_{\mathcal{G}} g(o_2)$ then o_1 is considered a better answer to the search problem than o_2. Notice that in a *Type 2* problem, a searcher that does not exhaust the whole of $\mathcal{R}$ cannot know whether an optimal object has been located (given that choice of $\mathcal{R}$), and hence has no reason to stop searching based on the quality of the objects it has located.

Our search algorithms represent potential solutions to the problem using instances of the data structure $\mathcal{D}$. At point t in time, the algorithm will have memory allocated for a finite set of these, and the values they contain will form a multiset, $\mathcal{C}_t \in \mathcal{M}(\mathcal{R})$. When we are not concerned with time, we will drop the subscript and simply use $\mathcal{C}$ to denote the current multiset of $\mathcal{R}$. In the discussion of landscapes to come, we will be concerned with $\mathcal{C}$ and with the methods used by the algorithm to change it. The algorithms employ *operators* to modify $\mathcal{C}$. In this dissertation, lower case Greek symbols will be used exclusively to refer to operators. In particular, ϕ (and less frequently ψ), will be reserved to indicate generic operators.

An operator is a function $\phi : \mathcal{M}(\mathcal{R}) \times \mathcal{M}(\mathcal{R}) \rightarrow [0..1]$. The value of $\phi(v, w) = p$ for $v, w \in \mathcal{M}(\mathcal{R})$ indicates that with probability p, v is transformed into w by a single application of the stochastic procedure represented by the operator ϕ. Thus, for any operator ϕ and $v \in \mathcal{M}(\mathcal{R})$,

$$\sum_{w \in \mathcal{M}(\mathcal{R})} \phi(v, w) = 1.$$

For convenience, I will talk of the "action" of an operator and mean the action of the computational procedure that the operator represents. Any modification made to $\mathcal{C}$ by the algorithm shall be considered to be the result of the application of a procedure associated with some operator. The series of applications of operators by an algorithm can be illustrated by

$$\mathcal{C}_0 \xrightarrow{\phi_1} \mathcal{C}_1 \xrightarrow{\phi_2} \mathcal{C}_2 \xrightarrow{\phi_3} \cdots.$$

The details of the workings of an operator have deliberately not been specified. As a result, operators are so broad in scope, that we may regard *every* method an algorithm uses to affect $\mathcal{C}$ as taking place through the action of an operator. Thus there are operators that (1) increase the size of $\mathcal{C}$ through the allocation of memory to serve as instances of $\mathcal{D}$, (2) reduce the size of $\mathcal{C}$ through freeing allocated memory, and (3) alter the contents of existing members of $\mathcal{C}$. Some operators may combine any of these three actions. Designating some action of the algorithm as being the result of an operator can be done quite arbitrarily. For instance, if we examine $\mathcal{C}_t$ and later examine $\mathcal{C}_{t+\delta}$, we can imagine that the algorithm has employed some operator that transformed $\mathcal{C}_t$ into $\mathcal{C}_{t+\delta}$ that took δ time units to complete. Depending on the value of δ, we may be talking of the effect of a single microcoded instruction in the CPU or of the action of the entire algorithm.

Clearly, some operators will have more effect on the result of the search than others. Though crucial for the algorithm's operation, an operator that increases the size of $\mathcal{C}$ by one (through memory allocation) will typically be uninteresting, as few algorithms deliberately make use of uninitialized memory. An initialization operator that sets the bits of an element of $\mathcal{C}$ may be somewhat more interesting. What we choose to consider an operator will depend on what we are interested in studying. In this dissertation, we will be concerned with operators such as mutation, crossover and selection in evolutionary algorithms. These represent one natural perspective on the working of these algorithms and reflect an interest in the importance of these operators. These operators are natural, in an informal sense, since they are typically implemented as separate procedures in the programs that implement the algorithms. Another natural perspective is to consider a generation of an evolutionary algorithm to be an operator. Such an operator usually includes some combination of the three finer-grained operators just mentioned. This is the perspective that has (so far) been adopted by researchers interested in understanding aspects of the genetic algorithm through viewing it as a Markov chain (see [52] for example). The set of operators of interest in an algorithm will typically be only a part of the algorithm. The algorithm must also make important decisions about which operators to apply and when, decide what elements of $\mathcal{C}$ should be acted on by operators, and decide when the search should be terminated.

Given an operator ϕ, the *ϕ-neighborhood* of $v \in \mathcal{M}(\mathcal{R})$, which we will denote by $N_\phi(v)$, is the set of elements of $\mathcal{M}(\mathcal{R})$ accessible from v via a single use of the operator. That is, $N_\phi(v) = \{w \in \mathcal{M}(\mathcal{R}) \mid \phi(v, w) > 0\}$. If $w \in N_\phi(v)$ we will say that w is a ϕ-neighbor of v. When the operator in question is understood, the terms "neighborhood" and "neighbor" will sometimes be used. However, it is important to keep in mind that this is an abbreviation. Two points that are neighbors under one operator may not be under another. Given $P \subseteq \mathcal{M}(\mathcal{R})$, we will use $N_\phi(P)$ to represent the set of elements of $\mathcal{M}(\mathcal{R}) - P$ that neighbor an element of P. That is,

$$N_\phi(P) = \{w \in \mathcal{M}(\mathcal{R}) - P \mid \phi(v, w) > 0 \text{ and } v \in P\}.$$

A situation that we will often encounter arises when an operator ϕ replaces an element $v \in \mathcal{C}$ by choosing one of a subset of $N_\psi(v)$. That is, the job of ϕ is to replace v with one member of the neighbors of v as defined by a second operator ψ. An example will make this clearer. Suppose we have a hillclimbing algorithm; that $\mathcal{R}$ is the set of all binary strings of length n; and that $v \in \mathcal{R}$ is the current location of the hillclimbing search. Operator ψ flips a randomly chosen bit in a binary string. Operator ϕ uses ψ some number of times to generate a subset of $N_\psi(v)$. These are placed by ϕ into temporary locations in $\mathcal{C}$. When some number of neighbors of v under ψ have been generated, or when one is found that is suitable, ϕ replaces v with the selected neighbor. Another example of this sort of operator is that which effects a move in a simulated annealing algorithm [53]. It employs a mutation operator to generate neighbors of the current point until it finds an acceptable one and sets the memory corresponding to the current point to the value of the selected neighbor.

This brings us to the final aspect of search algorithms that we will consider before describing landscapes. When an algorithm makes a decision about the relative worth of a set of multisets of $\mathcal{R}$ (such as the neighbors generated in the example above), it must have some basis for this decision. Although the decision can be taken by simple use of a pseudo-random number generator, it is more common that the algorithm will have some way of computing the worth of an element of $\mathcal{M}(\mathcal{R})$. We will suppose the search algorithm has a function $f : \mathcal{M}(\mathcal{R}) \to \mathcal{F}$, for some set $\mathcal{F}$, and a partial order $>_\mathcal{F}$ over $\mathcal{F}$. If $v, w \in \mathcal{M}(\mathcal{R})$ and $f(v) >_\mathcal{F} f(w)$ then the multiset v will be considered in some sense better for the purposes of continuing the search than the multiset w. Examples of such decision-making include the use of selection in an evolutionary algorithms to produce one population from another, pruning methods in dynamic programming and the acceptance test in simulated annealing. In the first two of these examples, the algorithm is effectively making choices about multiple elements of $\mathcal{R}$, rather than about a single element of $\mathcal{R}$. This is the reason that the domain of f is $\mathcal{M}(\mathcal{R})$. We will call $\mathcal{F}$ the *fitness space.*

In all cases, the function f must be chosen by the searcher, it is not part of the problem. It might appear that when solving a *Type 2* search problem that when evaluating multisets of size one, the searcher would always choose $f = g$, $\mathcal{F} = \mathcal{G}$ and $>_{\mathcal{F}} = >_{\mathcal{G}}$. While this might be a very natural choice, it is still a choice. Suppose, for example, that the problem is to find a Hamiltonian circuit through a given graph whose edges are labeled with distances. A *Type 2* instance of this problem may specify that a circuit c_1 can be considered better than a circuit c_2 if the sum of the distances on the edges of c_1 is less than the sum on the edges of c_2. However, the searcher might believe that the search might be best approached by looking for circuits that form simple closed curves, and therefore aim to minimize the number of times a circuit crosses itself. Or the number of crossings may be combined with other information, such as the length of the circuit, to form another function that is used for search. Of course, the answer that the algorithm eventually produces will be evaluated only in terms of the function g, but that does *not* mean that the search has to be conducted using it. In simulated annealing and genetic algorithms, it is common to search using an f that incorporates g and some other function that gives penalizes illegal objects (see, for example, [54, 55]). On a *Type 1* search problem, it is more apparent that the searcher needs to devise some measure of worth, since no information is given at all. In this case, the function that is chosen has been called an *heuristic* function in Artificial Intelligence.

2.4. Landscapes

A landscape is dependent on five of the components of search and algorithms discussed in the previous section. We may write a landscape as

$$\mathcal{L} = (\mathcal{R}, \phi, f, \mathcal{F}, >_{\mathcal{F}}).$$

The components are, respectively, the representation space, an operator, the function $f : \mathcal{M}(\mathcal{R}) \to \mathcal{F}$, for some set $\mathcal{F}$, and a partial order $>_{\mathcal{F}}$ over $\mathcal{F}$. As emphasized in earlier sections, a landscape is a metaphor by which we hope to imagine some aspect of the behavior of an algorithm. That can be done by viewing the 5-tuple as defining a directed, labeled, graph $G_{\mathcal{L}} = (V, E)$ where $V \subseteq \mathcal{M}(\mathcal{R})$, $E \subseteq V \times V$ and $(v, w) \in E \iff \phi(v, w) > 0$. A vertex $v \in V$ will be labeled with $f(v)$. An edge (v, w) will be labeled with $\phi(v, w)$, the probability that the action associated with the operator ϕ produces w from v. Though a landscape is formally defined by a 5-tuple, we will also talk of a landscape as though it were the graph that arises from the 5-tuple.

The label $f(v)$ attached to a vertex v can be thought of as giving the "height" of that vertex. This is in keeping with the imagery we usually associate with landscapes. This value will often be referred to as the *fitness* of the multiset of $\mathcal{R}$ represented by v. The partial order, $>_{\mathcal{F}}$, is used to determine relative fitness (height). In many cases, $>_{\mathcal{F}}$ will actually be a total order. The out-degree of a vertex in a landscape graph will be the same as the size of the neighborhood of the corresponding multiset. When $\phi(v, w) = \phi(w, v)$ for all $v, w \in V$, we will consider the landscape graph as undirected and draw a single edge between v and w with the understanding that the edge is bidirected. It should be remembered that the vertices of the landscape graph correspond to multisets of elements from $\mathcal{R}$, not to single elements of $\mathcal{R}$. This is important, because it allows us to define landscape graphs for arbitrary operators, not just those that act on and produce a single element of $\mathcal{R}$ (e.g., mutation in a genetic algorithm). Landscapes in this model are well-defined no matter how many elements of $\mathcal{R}$ the operator acts on or produces, even zero. It will also be important to remember that each operator employed by a search algorithm helps create a landscape graph. Thus if an algorithm employs three operators, it can be thought of as traversing edges on three graphs.

2.5. Definitions and Special Cases

This section provides definitions for some of the terms that are commonly encountered in discussions of landscapes. In all of the following definitions, we assume that a landscape $\mathcal{L} = (\mathcal{R}, \phi, f, \mathcal{F}, >_{\mathcal{F}})$ is given. Some of the following terminology is most useful when $>_{\mathcal{F}}$ is a total order. For example, if $>_{\mathcal{F}}$ is not complete, given v and $w \in N_{\phi}(v)$, it may be that neither $v >_{\mathcal{F}} w$ nor $w >_{\mathcal{F}} v$ is true. In this case, we cannot decide if either vertex is a peak.

2.5.1. ϕ-neighborhood and ϕ-neighbor

These important terms have already been introduced as they were necessary for the presentation of the model. They are defined in §2.3(24).

2.5.2. ϕ-maximum or ϕ-peak

A *ϕ-maximum* or *ϕ-peak* is a vertex

$$v \in V \mid \forall w \in N_{\phi}(v), f(v) >_{\mathcal{F}} f(w).$$

In words, a vertex is a ϕ-peak if its fitness value is greater than the fitness values of all its neighbors under the operator ϕ. When we are not concerned with any operator in particular, we will simply talk of peaks. Notice that a peak is a vertex in a landscape, not a point in $\mathcal{R}$ or $\mathcal{O}$. In some cases, e.g., with mutation operators, the vertex will correspond to a single element of $\mathcal{R}$, but in others, e.g., with crossover operators, it will not. This definition allows every operator to have peaks, instead of reserving the notion for a single-change (see §2.6.1(31)) or mutation operator. Consequently, a vertex that is a peak from the point of view of one operator is not necessarily a peak from the point of view of another. This is the reason for calling it a *ϕ-peak* instead of merely a *peak*. This terminology is more awkward, but the point is an important one.

2.5.3. global-maximum

A *global-maximum* or *global-optimum* of a landscape is a vertex

$$v \in V \mid \forall w \in V, f(v) >_{\mathcal{F}} f(w).$$

That is, a vertex is a global-maximum if it is at least as fit as every other vertex. If a vertex has maximal fitness, then it will be a global maximum no matter who its neighbors are. For this reason, we can call a vertex a “global maximum” without an operator prefix. If a vertex is a global maximum under one operator, then it will also be under all other operators.

2.5.4. ϕ-local-maximum

A *ϕ-local-maximum* or *ϕ-local-optimum* is a ϕ-peak that is not a ϕ-global-maximum. In this case, unlike with global maxima, the ϕ prefix is important. A vertex v that is a ϕ-local-maximum will usually not be a ψ-local-maximum as, in general, $N_\phi(v) \neq N_\psi(v)$.

2.5.5. ϕ-plateau

A *ϕ-plateau* is a set of vertices

$$M \subseteq V, |M| > 1 : \forall\, v_0, v_n \in M,\ \exists v_1, \ldots, v_{n-1} \text{ with}$$

$$f(v_i) = f(v_{i+1}) \text{ and } v_{i+1} \in N_\phi(v_i) \quad \forall\, 0 \leq i < n.$$

This is a connected set of at least two vertices that all have the same fitness. It is possible to move using ϕ between any two vertices visiting only vertices with equal fitness. Our usual three-dimensional image of a plateau involves a flat area in which, for the most part, a step in any direction will not result in a drop in altitude. This is not the case in the above definition, and it is important to keep this in mind. For example, a ϕ-plateau may surround areas of exceptionally low fitness, as does the rim of a volcano. In such cases, there may be no points on the plateau that have the same fitness as all their neighbors. Defining a plateau to better fit our intuitions is difficult and besides, what we will most often be dealing with, at least in the chapters that follow, will be a ϕ-plateau as defined here.

2.5.6. ϕ-mesa

A ϕ-mesa is a ϕ-plateau, P, whose points have fitness k such that

$$\forall v \in N_\phi(P), f(v) <_{\mathcal{F}} k.$$

That is, a ϕ-mesa is a ϕ-plateau with the additional property that no point on the plateau has a neighbor of higher fitness. Such a set is a connected region of the landscape in which an algorithm that only moves to vertices of equal or higher fitness may wander indefinitely without making any improvement. A ϕ-mesa of size one is also a peak.

2.5.7. ϕ-saddle-region

A saddle point is a notion that is usually associated with continuous spaces. It is not immediately clear how to define a saddle *point* in a high-dimensional discrete space. If it really is to be a point and it is a saddle in the normal sense, then one definition would simply require the point to have at least one uphill neighbor and at least one downhill neighbor. As this will be true of most points, the flavor of a saddle point in a real space is lost. The term becomes practically useless since the only points that are not then saddle points are peaks (maximal or minimal).

A better definition is to use *ϕ-saddle-region* to refer to a ϕ-plateau that is not a ϕ-mesa. This is a region rather than a point. This definition also loses an important part of the flavor of a saddle point, but at least it is useful and the name change highlights the loss.

2.5.8. ϕ-basin-of-attraction

The ϕ-basin-of-attraction of a vertex v_n is the set of vertices

$$\begin{aligned} B_\phi(v_n) = \{\, v_0 \in V \mid \exists v_1, \ldots, v_{n-1} \text{ with} \\ v_{i+1} \in N_\phi(v_i) \quad \forall\, 0 \le i < n \,\}. \end{aligned}$$

Thus the basin of attraction of a vertex v is the set of vertices from which v may be reached using ϕ. Notice that $w \in B_\phi(v) \not\Rightarrow v \in B_\phi(w)$.

2.5.9. Fixed Cardinality Operators

Given $\phi : \mathcal{M}_k(\mathcal{R}) \times \mathcal{M}_l(\mathcal{R}) \to [0..1]$ for a fixed k and l, then we shall say that ϕ is a *fixed cardinality* operator. Otherwise, ϕ will be said to be of *variable cardinality*. We will use $\phi_{k \to l}$ to represent the set of all fixed cardinality operators with domain $\mathcal{M}_k(\mathcal{R}) \times \mathcal{M}_l(\mathcal{R})$. Fixed cardinality operators will be treated so commonly in what follows, that an unqualified use of the word "operator" implies that the operator has fixed cardinality.

2.5.10. Walkable Operators

Fixed cardinality operators that are members of $\phi_{k \to k}$ will be called *walkable*. If an operator is walkable, the output of the action corresponding to the operator can be used as its next input. For example, the common forms of mutation are always walkable. Any form of crossover that takes two parents and produces two children is walkable. Crossover that produces one child from two parents is not walkable.

2.5.11. Symmetric Operators

An operator ϕ will be called *symmetric* if $\phi(v,w) = \phi(w,v)$ for all vertices $v, w \in V$. That is, an operator is symmetric if the probability of it producing v from w is the same as the probability of it producing w from v. Notice that if this condition is true, then so are the implications:

$$w \in N_\phi(v) \Rightarrow v \in N_\phi(w) \quad \text{and} \quad w \in B_\phi(v) \Rightarrow v \in B_\phi(w).$$

Hence a symmetric operator is necessarily walkable. Most common operators in evolutionary algorithms are symmetric. As mentioned in §2.4(25), if an operator is symmetric it will be convenient to consider $G_\mathcal{L}$ as undirected, in which case edges may be considered bidirectional and will not be drawn with arrowheads.

2.5.12. ϕ-connected-components

If an operator is symmetric, we will often talk about the ϕ-connected-components of a landscape graph. Two vertices $v, w \in V$ are ϕ-connected in $G_\mathcal{L}$ if there is at least one path between them (irrespective of edge directions). Following the definitions of connectedness in §2.2.2(18) leads naturally to ϕ-connected landscapes. The operator associated with a connected landscape is necessarily walkable, but the converse is not true, as will be demonstrated in Chapter 3. If

$$\phi(v,w) = \phi(v,x) \quad \forall v \in V \text{ and } \forall w, x \in N_\phi(v)$$

we will usually not label the edges of $G_\mathcal{L}$ with the transition probabilities. Figure 1 illustrates a situation where transition probabilities are not equal.

2.5.13. Natural Landscapes

The landscape induced by a fixed cardinality operator will be called *natural* if the operator is symmetric and an element of $\phi_{1\to 1}$. Most mutation operators generate natural landscapes. The name is chosen as this is what people usually refer to as a landscape.

2.6. Four Operator Classes and Their Landscapes

This section presents four familiar operators, single-change, mutation, crossover and selection, and briefly describes some properties of the landscapes they generate. These

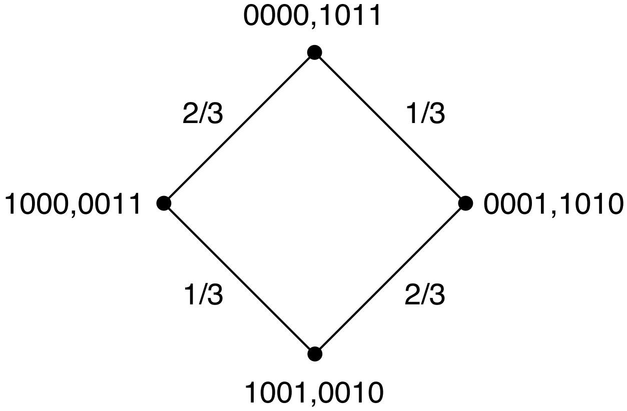


Figure 1. A fragment of the one–point crossover landscape for binary strings of length 4, showing unequal transition probabilities. The operator is symmetric, so the edges of the graph should be considered bidirectional.

operators will be treated so commonly in the remainder of this dissertation that we will give them special names. Three of the four classes of operators most commonly encountered in evolutionary algorithms rely on the objects in $\mathcal{R}$ being composed of some (not necessarily fixed) number of components. For example, a bit string has some number of bits, an n-tuple of real numbers consists of n individual reals and the parse tree representing a LISP S-expression consists of some number of internal nodes and leaves. These components can typically be modified individually or in subsets, possibly according to some constraints.

2.6.1. Single-change Operators

The simplest class of operators I will consider, a subset of $\phi_{1\to 1}$, act on and produce a single element of $\mathcal{R}$. These operators randomly choose a single component of their input and modify it in some way. The simplest example of this sort of operator is the operator that flips a bit chosen uniformly at random in a bit string, which we will denote by β. Note that the outcome of this operator *always* differs in exactly one component from the input. If the elements of $\mathcal{R}$ have n binary components, β induces a graph that is a hypercube of dimension n. Each edge in the graph has a transition probability of $1/n$. Figure 2 shows the landscape generated by this operator on binary strings of length three.

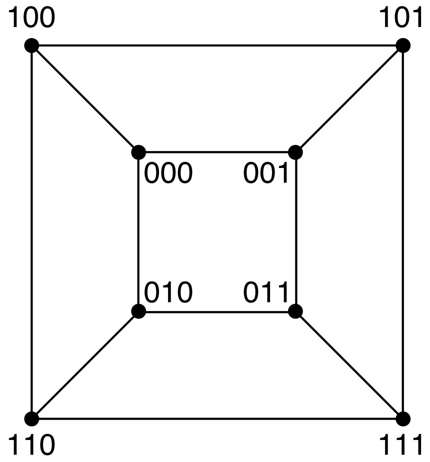


Figure 2. The landscape for the bit-flipping operator (β) on binary strings of length three. Edges are bidirectional and each has probability one-third.

2.6.2. Mutational Operators

The mutational operators, also a subset of $\phi_{1\to 1}$, modify each component of their input with a given, typically small, probability. Thus a mutational operator's output is a modification of its input according to some probability distribution. The outcome may be identical to the input, it may contain minor changes or may even be different from the input in every component. We will denote a generic mutational operator by μ. If we again consider binary strings of length n and assume that each bit is flipped with a fixed probability p, then the graph induced is the complete graph K_{2^n}, augmented with a loop at each vertex. An edge in the graph between vertices that represent binary strings whose Hamming distance is d will be labeled with a probability $p^d q^{n-d}$, where $q = 1 - p$. Notice that the structure induced by this operator is quite different from the one generated by the conceptually similar bit-flipping operator. Figure 3 shows the mutation landscape for binary strings of length three. The operators in both of these classes are walkable. The hypercube induced by β operator will, of course, be a subgraph of the graph generated by μ. If a mutation operator induces a complete graph (possibly augmented with loops), then this graph will always contain exactly one mesa, whose points will have maximal fitness. If no two points in the space have the same fitness, this mesa will be of size one and hence a peak.

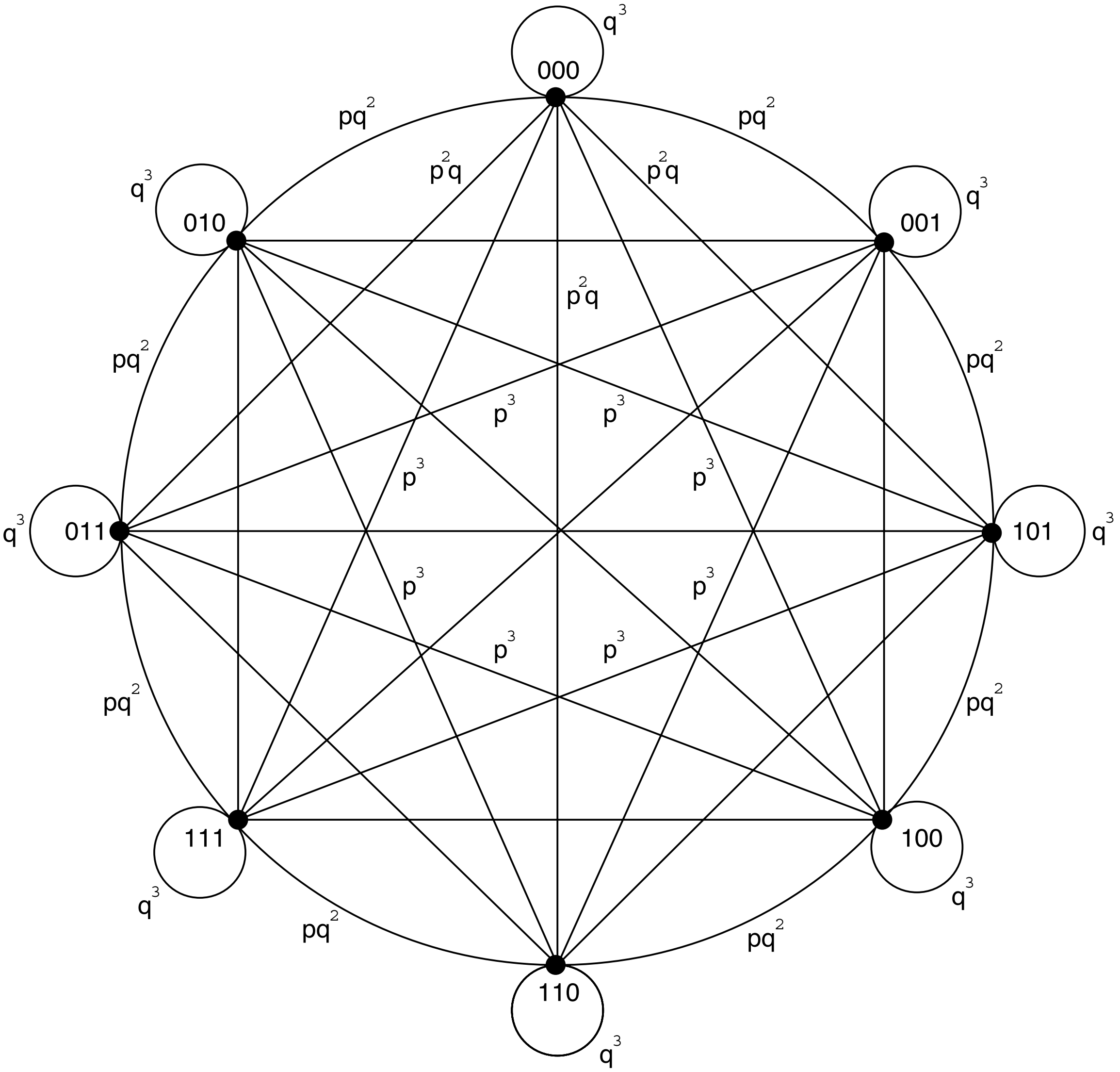


Figure 3. The mutation landscape for binary strings of length three. The mutation probability is p, and $q = 1 - p$. Some edge probabilities are omitted. Edges are bidirectional.

2.6.3. Crossover Operators

The third class, the crossover operators, typically act on an element of $\mathcal{R}^2$. Crossover operators combine the components from their inputs to produce their output. As an example, in one form of crossover in an evolution strategy, the operator receives two n-tuples of real numbers as input and produces a single n-tuple whose elements are the averages of the corresponding components of the inputs [56]. In GAs, the original form of crossover, as described by Holland [9], was *single-point* crossover, which has become more frequently known as *one-point* crossover. There are many forms of crossover for representation spaces such as S-expressions [14, 57, 15], n-tuples of reals [56], permutations of integers [58, 59] and fixed length strings over some alphabet [5]. Figure 4 shows the landscape generated by one-point crossover on binary strings of length three. A generic crossover operator will be denoted by χ. To indicate that a crossover operator acts on k elements of $\mathcal{R}$ and produces l elements of $\mathcal{R}$, a superscript will be added: $\chi^{k \to l}$. A subscript will be used to indicate the common types of crossover: 1 for one-point, 2 for two-point etc., and u for uniform crossover. Thus $\chi_1^{2 \to 2}$ is the crossover operator that produces two offspring from two parents using one-point crossover.

2.6.4. Selection Operators

The fourth class is the selection operators. A selection operator acts on a population of individuals (i.e., an element of $\mathcal{M}(\mathcal{R})$) and produces a population. In GAs and their derivatives, selection produces a population whose size is the same as the size of the input population. In some ESs, selection produces a smaller population. For example, in a $(\mu+\lambda)-$ES, the selection operator is a member of $\phi_{\mu+\lambda \to \mu}$. The selection operators exercise two generalities of the model that have not yet been shown to correspond to anything. First, if we regard selection as being an operator that acts on a population of any size, then it is a variable cardinality operator. The second is that selection operators produce landscapes that are not symmetric. Typically, the higher $\phi(v, w)$ is, the lower $\phi(w, v)$ will be, since selection, almost by definition, tends to increase the number of copies of the fittest individuals and decrease the numbers of the less fit. As a result, the above probabilities would very likely only be equal for populations with highly uniform fitness distributions.

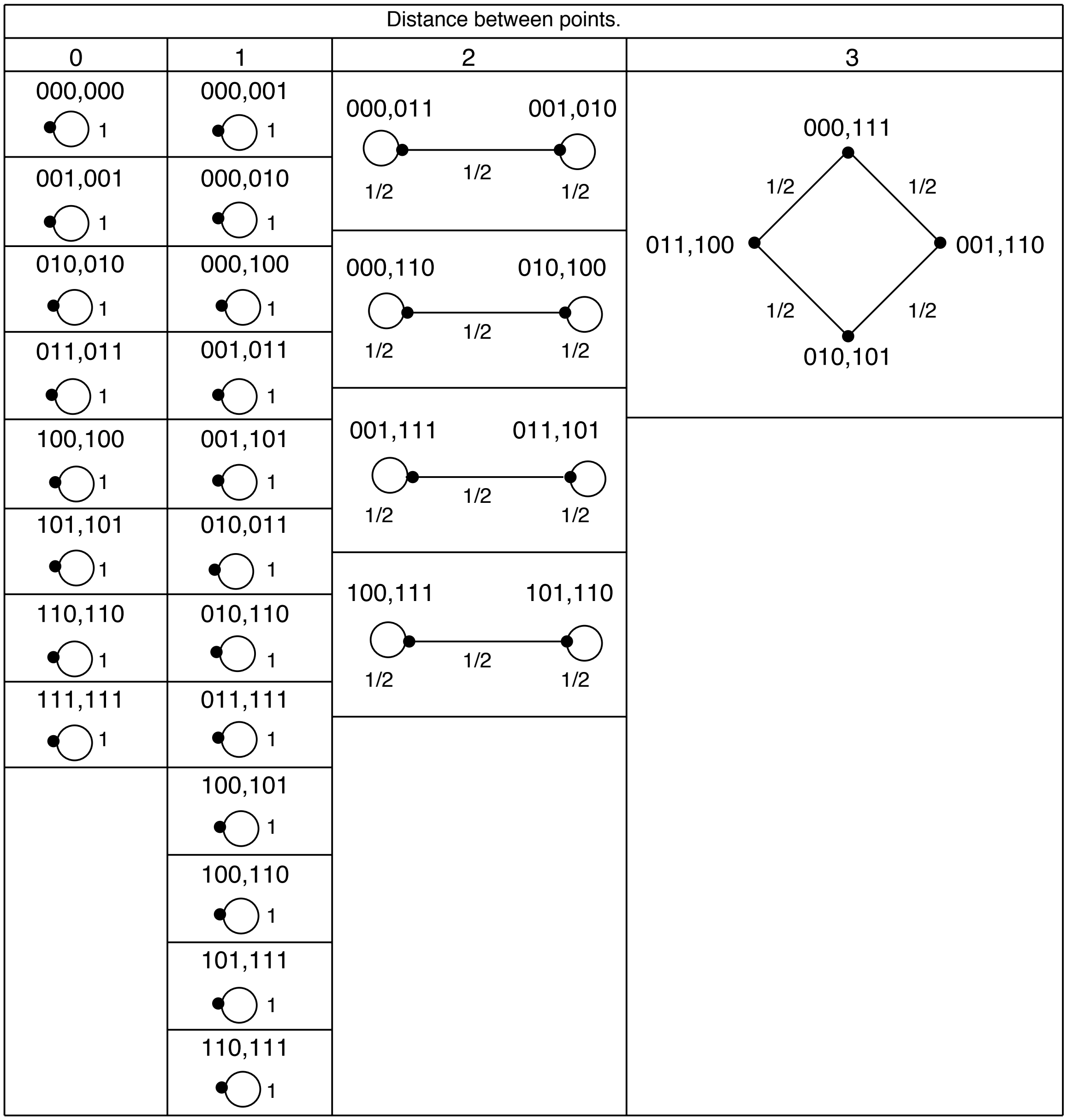


Figure 4. The one-point crossover landscape for binary strings of length three. The crossover operator, $\chi_1^{2\to 2}$, produces two offspring from two parents. Edges are bidirectional.

2.7. Consequences of the Model

The model presented above has a number of consequences that are not found in other landscape models. Two of these may seem particularly strange. First, as described above, a landscape may not be *walkable*. As mentioned in §2.5(26), an example is the landscape induced by any form of crossover that produces one child from two parents. Such an operator cannot be used to conduct a walk on the landscape as the output of the operator cannot be used as the next input.

Second, landscapes may not be connected. A simple example is the landscape induced by a (non-GP) crossover operator that produces two children from two parents. Consider a vertex of the landscape that corresponds to two points of $\mathcal{R}$ that are identical. The vertex will be connected to itself (with probability 1), and nothing else. A more specific and less trivial example of a landscape that is not connected is seen by considering the vertex $(011, 010)$ of V where $\mathcal{R} = \{0,1\}^3$. Clearly no form of crossover can transform this input to a pair of points either of which begins with a one, e.g., $(100, 000)$. Equally clearly, no composition of crossovers can accomplish this either. Thus $(011, 010)$ is not connected to any vertex in the landscape which contains a member of $\mathcal{R}$ that starts with a 1 (in fact, this vertex is also connected only to itself). Several complete crossover landscapes are presented in Chapter 3.

The possibility that landscapes may not be connected means that on these landscapes there is no general notion of distance between landscape vertices. This does not mean that a metric cannot be defined, just that there may be no natural one (such as the length of the shortest path between the vertices concerned). The model does not require a distance metric to exist, though the absence of one might make certain statistics meaningless or impossible to compute. Within a connected component of a landscape, one can always use the length of the shortest path between two vertices as a distance metric. The subject of distance metrics and dimensionality is addressed in §2.9(39).

Because a landscape may not be walkable, general statistics describing properties of landscapes may be restricted to using the information gained from repeated single applications of the operator that generated the landscape. This sort of statistic was employed by Manderick et al., even though they were considering a walkable landscape [60].

2.8. Implications for Genetic Algorithms

An important consequence of this model of landscapes is that every operator used by an algorithm defines its own landscape. For this reason, genetic algorithms can be seen as operating on several landscapes; a mutation landscape, a crossover landscape and a selection landscape. These landscapes will have different characteristics for different problems, and can be studied independently. I will refer to this as the "one operator, one landscape" view.

When one hears people talk of "the landscape," the object of reference is usually the mutation landscape or, more commonly, the hypercube landscape that β induces. Despite the fact that crossover may be employed more frequently than mutation in a GA, it has been assigned a supporting role in the landscape structure defined by mutation. To say that crossover is making large jumps on the landscape (the mutation landscape is implied), is to examine crossover outside its own context. What if the algorithm involved does not even use mutation? What relevance does the mutation landscape have then for the algorithm or for crossover? It makes as much sense to say that mutation is making large jumps on the crossover landscape. Though it is obviously important to consider the effect of using multiple operators in an algorithm, it is also important that when we assess an operator in isolation that we do it in light of the landscape structure it defines.

As was mentioned in §2.1(13), the notion of a peak cannot be divorced from the notion of neighborhood. A vertex in $G_{\mathcal{L}}$ can be a peak under one operator and not under another. The mutation landscape has peaks, the crossover landscape has peaks and the selection landscape has peaks. To regard peaks as something defined only in terms of mutation is simply biased.

Another popular notion is that GAs are somehow making a walk on a landscape. I would argue that it is only rarely that such a claim can be supported. A GA that employs crossover that produces one child from two parents and no mutation whatsoever could hardly be said to be walking. This example illustrates the fact that a GA does not require its operators to generate walkable landscapes. It does not follow that when they do, the GA is walking on them. My view of the basic cycle of a GA is of a process that is making moves on three landscapes. Each member of the population corresponds to a vertex in the mutation graph. Some of these are moved, depending on the mutation probability. Then the population members are somehow paired, which defines a set of vertices on a crossover landscape. Many of these vertices make a move on this landscape, depending on the crossover probability. Finally, the entire population corresponds to a point on the selection landscape and a step is taken there before the cycle repeats. This process is depicted in Figure 5. Under this view, the GA is taking single steps on the various landscapes but cannot be said to be walking on any of them. In some cases, obviously, a point may survive many such cycles and take several steps on the mutation landscape. In other cases the population might be highly converged, in which case crossover and mutation will be producing little change and it could be argued that some or even most of the population is walking on the mutation landscape or that the entire population is walking on the selection landscape. But these cases are the exception rather than the norm.

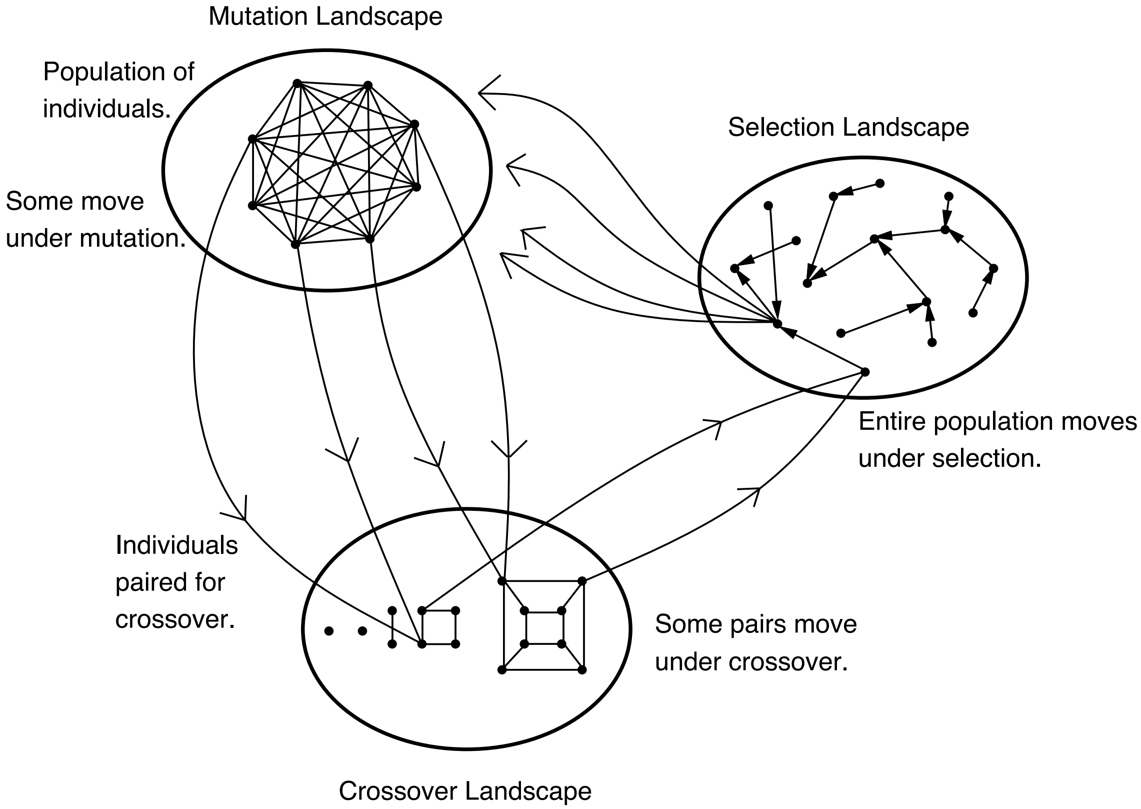

Mutation Landscape
Population of individuals.
Some move under mutation.
Selection Landscape
Entire population moves under selection.
Individuals paired for crossover.
Some pairs move under crossover.
Crossover Landscape

Figure 5. A simplified view of a GA operating on three landscapes. The landscape graphs are idealizations of far larger structures, and self-loops in the graph, created when an operator's input and output are identical, have been omitted. The GA is seen as taking some steps on the mutation landscape, then pairing individuals (probably according to fitness) thereby forming vertices on the crossover landscape upon which moves are made before the entire population is gathered into a vertex on the selection landscape where a step is taken. Finally, the population is decomposed into individuals which again correspond to vertices on the mutation landscape.

2.9. Advantages of the Model

Apart from fact that the model makes it possible to use the landscape metaphor in an unambiguous way, there are several other advantages that deserve mention:

- The model establishes a point of contact with the search algorithms of AI and OR. In all cases the underlying structures being searched are graphs. This relationship is explored more fully in Chapter 5.

- The model does not make the assumptions that prevent other notions of landscape from being more widely used. For example, the landscape does not need some fixed dimensionality, nor does it need a distance metric between the objects that compose the landscape. For these reasons, it is possible to view many AI problems as being problems on landscapes of this type. For instance, one could conceive of a landscape for chess or Rubik's cube. The navigational task on these landscapes may differ, or the operators may be complex, but the underlying structures are the same. The model is also useful within the field of evolutionary computing. It applies as well to GP as to GAs. Statistics that can be calculated for a landscape in one paradigm can be calculated in exactly the same way for another. The model also provides a framework for thinking about HC, SA, ESs and EP.

- The model invites a point of view that seems uncommon in the field of evolutionary computation, though not in AI. This is a view of search as navigation and structure. Once we view search in this way and identify the various components present in a GA, it is natural to ask questions about them. This division and the recombination it makes possible are examined in detail in Chapter 3.

- The "one operator, one landscape" view reveals the very different landscapes that are constructed by various operators. This invites statistical analysis of the landscapes, as will be seen in Chapter 5 and as has been done in [60, 61, 62, 63, 64]. Such analysis has the advantage of being independent of any particular navigation strategy. For this reason, it may be possible to demonstrate that a particular operator creates difficult (in some sense) landscapes for some problem types. Such results might go a long way towards resolving debates on the virtues of certain operators. Statistics such as these would be very useful as indicators of potential difficulty (or ease) of a problem for an operator.

2.10. Limitations of the Model

It can be argued that the model is not particularly useful in situations where the nature of the operators changes in the course of search. For example, the change of mutation vector for an individual in ES and EP, the change of edge probabilities when the temperature falls in simulated annealing or after inversion in a GA, or where the mapping between objects and representation space is changed, e.g., in dynamic parameter encoding [49], delta coding and delta folding [48]. There is some truth to this argument. However, if one ceases to regard a search structure as something necessarily fixed for all time, the model is still potentially useful. For example, a statistic, say correlation length, might be computed for the landscapes generated by a range of different temperature settings in a simulated annealing problem. This might provide useful information about when the search could be expected to make good progress (thus guiding the choice of cooling schedule) and it might prompt a comparison with a hill climber or other algorithm on one of the landscapes. These situations make the landscape something of a moving target, but the targets can be studied individually.

More generally, an algorithm might change many aspects of its behavior, for example the fitness function or the representation space, mid-run, and thereby shift its attention to new landscapes. This might be done very frequently. In these cases, the landscapes model of this dissertation holds that each such change produces potentially new landscape structures and that these can and should be studied independently. This is not a claim that the family of landscapes used by an algorithm should not be studied as a whole, just that it is possible to study the components in isolation, that this will be a simpler task and that it is worthwhile.

The fact that the model does not *require* a distance metric is not a limitation. There is nothing in the model that prevents the definition of a distance metric on $\mathcal{R}$ or V, and if this proves useful, it should be taken advantage of. In cases where the landscape is fully connected, there is always a natural definition of distance, and this can be used to compute such things as correlation lengths [65]. In other cases there may be no useful definition of distance. That the model does not provide one is not a shortcoming. It cannot be denied that the algorithm is making moves on the graph defined by the operator. A landscape model that provides a concrete, well-defined graph which can be studied, is far better than nothing.

2.11. Operators and Representation

The aim of this section is to make it clear that representation and operators are not independent and to illustrate how they interact. The choice of one constrains the possible choices of the other. To some this will be obvious, but others (I was one) may not have realized how tightly bound the two are. A choice of $\mathcal{D}$ is also a choice to use operators which act on that data structure.

There is an important tradeoff in these choices. Typically, the more intuition that has gone into the choice of $\mathcal{O}$, the more difficult it will be to design operators that act on and produce members of $\mathcal{O}$. This is related to the problems with maintaining legal objects which are addressed in §2.12(41). Here I am more concerned with the tradeoff between the recognition of important classes of objects and the construction of operators to move between them.

An example that will be relevant to experiments of this dissertation concerns the representation of Turing machines. A simple representation will ignore the fact that there are many symmetries amongst Turing machines that, if removed, would result in far smaller search spaces. The simple solution is to ignore these symmetries, which allows the use of simple operators. Once the symmetries are noted, they can be excluded from consideration by adopting some canonical Turing machine representation, but it may be difficult to construct operators that produce canonical machines from other canonical machines. This particular example is dealt with in greater detail in §3.8.5.1(61). Unfortunately, reducing the size of the space that is being searched does not necessarily make the search problem simpler (see, for example, [66, 67]). The tradeoff between representations and operators will be encountered again in §5.3.1(124).

2.12. Objects and Representation

There is much that can be said about the choices of $\mathcal{O}$ and $\mathcal{R}$ and the relation, Γ, between them. As mentioned in §2.3(20) a choice of representation is a choice of data structure and this choice leads immediately to a set $\mathcal{R}$ whose members correspond to the various ways the bits in a data structure representing an object can be filled. Note that this does *not* restrict attention to fixed length data structures.

The problem of course is that there may be many ways of filling a data structure that do not result in the representation of an object in $\mathcal{O}$. A similar problem exists when the problem statement involves constraints on the form of the solution. The representation space that is chosen may include elements that correspond to objects that violate constraints. These situations are very common in evolutionary algorithms, and a number of ways of dealing with them have been adopted. They arise in even the simplest GA applications (for example, using 2 bits to represent 3 objects), in algorithms that manipulate permutations (where crossover can easily produce a non-permutation), and with floating point representations. There have been several approaches to dealing with these problems:

- Probably the most common solution is to build special operators that produce legal representations from other legal representations. This is a common approach when manipulating permutations of integers. Operators for this include Cycle Crossover [58], Order Crossover [58], Partially Matched Crossover [68], Edge Recombination [59, 69, 70], the crossover of Gorges-Schleuter [71], Maximal Preservative Crossover [72], Strategic Edge Recombination [73], and Generalized N-point Crossover [67]. Michalewicz describes special-purpose operators designed to stay within a feasible region of $\mathbb{R}^n$ given by linear constraints [74].
- Another solution is to allow these illegal representations but to penalize them somehow to encourage the algorithm to avoid these regions of the space, possibly with increasing probability over time [5, 54, 75, 76, 77, 78, 79].
- Davis and Steenstrup [80] suggest that the problem can be dealt with by "decoding" illegal individuals before evaluating them. They do not give an example, but claim the procedure is often computationally intensive.
- Another approach is to allow the operators to construct illegal representations but to repair the result [81, 82], a procedure called "forcing" by Nakano et al. [83].
- If the size of the subset of $\mathcal{R}$ that is legal is not too small, a solution is to generate individuals repeatedly until a legal one is found.
- A recent elegant solution, proposed by Bean, adopts a new representation of a permutation that allows traditional operators to be used [84].

This dissertation does not attempt to treat representational issues such as the above, though this aspect of search is as important as any other (see §2.13(42)).

2.13. Origins and Choices

This section discusses the origins of some of the components of search algorithms and landscapes. The importance of the choice of $\mathcal{O}$, the objects that are being considered as possible solutions to the search, was illustrated in §1.3(5) with the eight queens problem. This is by no means an isolated example. This choice might not seem part of the search algorithm, but it is simple to view it as such. Perhaps the reason why this aspect of search is not so considered is that we typically have no idea how such a choice could be performed by a machine. The choice requires experience, insight and creativity. It is possible for an algorithm to make this choice and to explore various answers during the course of a search, but currently we do not know how the process works well enough to design an algorithm to perform it. If this component of search eventually falls into the domain of the computation, artificial intelligence will have taken a significant step.

The choice of the representation space, $\mathcal{R}$, has similar importance and is also typically made by the programmer, not the program. An important issue is that of what I call *over-representation* and *under-representation*. Over-representation occurs when many elements of $\mathcal{R}$ are interpreted (according to Γ^{-1}) as corresponding to the same element of $\mathcal{O}$. From a problem solving point of view, such a choice is strictly redundant, but it may have advantages that we do not yet fully appreciate [66, 67]. The mapping between RNA primary and secondary structure is highly redundant in this sense. GP has a similar flavor; every S-expression is a member of an infinitely large class of S-expressions that are all functionally equivalent.

Under-representation is very common in problem solving. A dramatic and elegant example is Kanerva's Sparse Distributed Memory in which an extremely large address space (e.g., of 2^{1000} locations) is represented in a conventional-sized memory [41]. As mentioned earlier, an object space that involves an infinite set must be under-represented at any one point in time. Genetic algorithms have traditionally under-represented real intervals via a discretization of the interval indexed by the binary value of a bit string. Using a floating point representation also under-represents the reals, but in a far less drastic manner. Work on issues of representation, particularly changing representation, can be found in [47, 48, 49, 50, 51, 85].

A representational choice that is common in AI and OR has the elements of $\mathcal{R}$ represent *partial* objects from $\mathcal{O}$. A partial object can also be viewed as representing the class of objects that in some sense contain the partial object. For example, a sub-tour in a graph can be thought of as a partial tour, but also as a representative of all those complete tours that include the sub-tour. This form of representation is the basis of the "split-and-prune" [31] paradigm of OR and it underlies all of the many variants of the branch-and-bound algorithm. A *schema* in a GA is also representative of a class of objects (in this case binary strings), though the GA does not manipulate schemata explicitly,[3] though each binary string of length n is an instance of 3^n schemata and thus operations on individuals can be thought of as operations on samples from many schemata. This was described as *intrinsic parallelism* by Holland in [9] but has become more widely known as *implicit parallelism*.

The final choice I want to mention is that of operators. Every instance of every search problem has a perfect operator. If the operator could be found (which of course it cannot), very little else would matter. A random walk search strategy would locate a global optimum in at most two evaluations. This operator would build a graph similar to that shown in Figure 6. Paradoxically, an operator that would be even harder to construct would result in a difficult problem. This operator connects each vertex directly to the vertex that has the next highest fitness. This is essentially the landscape that was produced by Horn et al. in [86], though that landscape was produced by combining a very specific fitness function with an ordinary operator and this is produced by combining a very specific operator with an ordinary fitness function. An example is shown in Figure 7. The importance of the choice of fitness function is discussed at length in the second half of Chapter 5.

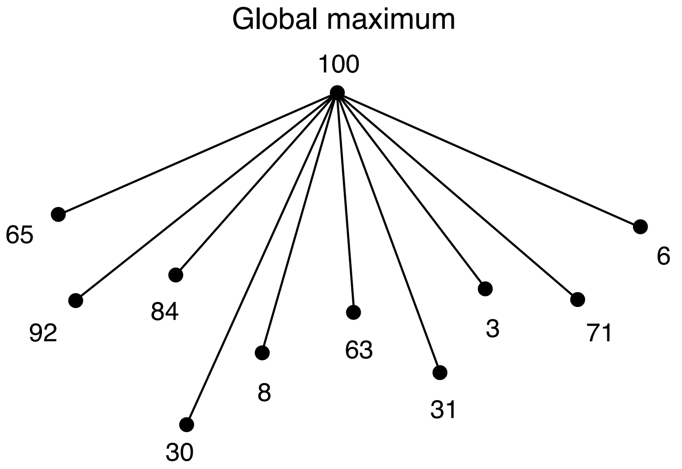


Figure 6. The landscape generated by a perfect operator. Every point in the space, except the global optimum is connected directly to the global optimum. Vertices are labeled with fitness values. Any search strategy using this operator will find the global optimum quite rapidly.

[3] This is not strictly true if a binary string with no don't care symbols is regarded as a schema, which it is sensible to do. A more accurate statement would assert that schemata of order greater than zero are not explicitly manipulated by the GA.

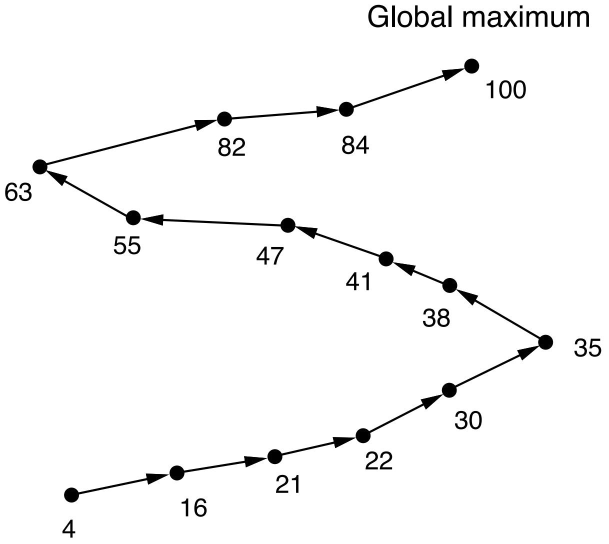


Figure 7. The landscape generated by a not-so-perfect operator. Every point in the space, except the global optimum is connected directly to point with next highest fitness. Vertices are labeled with possible fitness values. Any search strategy that starts at a randomly chosen vertex and thereafter uses this operator alone to generate new possible solutions will explore an average of half the space on each successful search.

The point of these remarks and illustrations is simply to emphasize that in all cases these choices have to be made and that the choices that are made may have a tremendous impact on the difficulty of a problem. I am not suggesting that, given a search problem, we should search for the perfect operator or fitness function. In every case, the choices that are made determine a landscape as described by the current model. This suggests instead that we should try to develop techniques to study these structures, and attempt to apply what we learn to the problem of deciding whether one choice in their construction appears better than another. The landscape model that underlies these structures is in all cases a graph, so a technique developed to study an aspect of one landscape can be automatically applied to many others.

2.14. The Usefulness of the Landscape Metaphor

The term “landscape” has something powerfully seductive about it. The imagery it evokes is so appealing, that further thought can be completely suspended. An important question to ask is why we would want to use such a term. The answer is presumably that we hope to use the imagery (e.g., peaks, ridges, valleys etc.) to enhance our understanding of some process, to develop new ideas for exploring spaces and to stimulate questions about processes operating on these structures. All of this tends to rely rather heavily on the simple properties that we see in physical three dimensional landscapes. It is not clear just how many of the ideas scale up to landscapes with tens or thousands of dimensions. It is quite possible that the simplicity and beauty of the metaphor is actually damaging in some instances, for example by diverting attention from the actual process or by suggesting appealing, simple and incorrect explanations. Many of these potential problems are summarized by Provine [87, pp. 307–317], which should be required reading for people interested in employing the metaphor. Wright's response to Provine's criticism is that his landscape diagrams were intended as a convenient, but simplistic, representation of a complex process in a high-dimensional space [37], and were never intended for mathematical use.

As outlined in §2.1, similar problems exist in evolutionary computation. Given this, it is worth asking whether it is better to abandon the term or to use it and try to be more precise about what is actually meant. There is something to be said for abandoning it. After all, in virtually every formulation, a landscape can be regarded as a graph. On the other hand, it seems unlikely that the term will just go away. In addition, the metaphor, however distant it may sometimes be from reality, *has* given rise to new ideas and intuitions. I have chosen to adopt the term, with the hope that it will lessen, rather than increase, the vagueness with which it is applied.

2.15. Conclusion

This chapter presented a general model of landscapes and an overview of its consequences, advantages, limitations and relevance to evolutionary algorithms. The model views a landscape as a directed graph whose edges and vertices are labeled. It was argued that the operators in evolutionary algorithms each generate a landscape, that these landscapes have differing qualities, and that each can and should be studied in its own right. Thus most evolutionary algorithms are seen as operating on multiple landscapes. Defining a landscape as a graph establishes a contact with search algorithms from artificial intelligence and operations research, many of which are explicitly designed to search labeled graphs. The model advocates a view of search as composed of navigation and structure, with the structure provided by landscapes. It is argued that the statistical properties of landscapes can be studied independently of navigation strategies. The relationship between this model and other work on landscapes is dealt with in detail in Chapter 6. Most of the issues touched on in this chapter will be encountered in the chapters that follow.

CHAPTER 3

Crossover, Macromutation, and Population-based Search

3.1. Introduction

The features of a GA that distinguish it most from other search methods are its use of a population and a crossover operator. In a GA, crossover is the major mechanism of communication between individuals in a population. The usefulness of crossover, and therefore, to some extent, of maintaining a population, has been a contentious issue in research on GAs (see, for example, [29, 88, 89, 90, 91, 92]). The standard method of determining whether crossover is useful to a GA involves comparing a GA with crossover to a GA without crossover. If the GA with crossover outperforms the GA without crossover, this is taken as evidence that crossover is useful (i.e., that it is facilitating the exchange of building blocks between individuals). Therefore, the maintenance of a population and the use of crossover seem justified because these are clearly an aid to the algorithm. This chapter argues that this conclusion is not justified and gives an alternate method for assessing the utility of crossover.

A distinction is made between the *idea* of crossover and the *mechanics* by which crossover operators attempt to implement this idea. It is shown that even in the absence of the idea, the mechanics alone can be effectively used for search. A GA with no crossover has neither the idea nor the mechanics of crossover, and if it is outperformed by a normal GA, it does not follow that the idea of crossover is therefore useful to the normal GA. An algorithm is presented that illustrates the power of the mechanics of crossover. This suggests a simple new comparison for determining whether crossover and a population are providing some additional benefit to the GA over simpler algorithms that do not use crossover or maintain a population. This comparison is between a normal GA and a GA that uses *random crossover*. Using this, it possible to see what advantage the GA is gaining from the idea and the mechanics of crossover over and above what it could be gaining from the mechanics alone. In some cases, when well-defined building blocks are not present, the GA may actually perform *worse* with normal crossover than a GA with random crossover because of its use of a population. This test gives an indication of the existence of building blocks that are exploitable by a GA with a given crossover operator.

3.2. Search as Navigation and Structure

The model of landscapes presented in Chapter 2 maintains that every operator used in a GA creates its own landscape, and advocates a view of search as a process of navigation upon these landscapes. This division into structure and navigation invites the construction of new algorithms. A potentially new search algorithm may be constructed from two existing search algorithms by combining the navigation strategy of one algorithm with the landscape structure(s) from the other. This observation prompted consideration of combinations involving the representation used in genetic programming and navigation strategies of hillclimbing and simulated annealing by O'Reilly and Oppacher [93].

This process is possible because navigation strategies are designed to search graphs and landscapes are graphs. This implies that the pieces will be interchangeable provided the graph satisfies any assumptions made by the navigation strategy (e.g., that it is a tree). The resulting search algorithm may be interesting in its own right (for instance because it performs better than one or both of the originals in some circumstances) or it may be interesting because it throws light on one or both of the original algorithms. In this chapter the combination of the navigation strategy of a simple hillclimber with the landscape structure of crossover is examined in detail. The resulting *crossover hillclimbing* (CH) algorithm is interesting for both of these reasons—it outperforms both the GA and the hillclimber, and, an investigation into the power of the algorithm leads to a clearer understanding of the dual role of the crossover operator in a GA. This combination of navigation strategy and operator has also been investigated, in the context of genetic programming, by O'Reilly and Oppacher [94].

Navigation strategy has deliberately not been well defined. This allows us to apply the spirit of this division (into navigation strategy and structure) to a very wide range of algorithms, without requiring a specific definition of what a navigation strategy actually is. Landscape graphs on which the algorithms make moves are well defined in §2.4(25). We will define navigation strategy as any part of a search algorithm that is not a landscape. For example, consider a simple hillclimbing algorithm that operates on binary strings using the bit-flipping operator, β, described in §2.6.1(31). Each of the vertices in the landscape graph of β corresponds to a binary string, and each has an associated fitness. If we suppose that the algorithm starts operation at a randomly chosen vertex, v, to which vertex of $N_\beta(v)$ does it move next? This decision is made by the navigation strategy. Notice that we have assumed the algorithm has made the choice to use the operator to generate a new binary string. This choice is part of the navigation strategy, though in such a simple algorithm there is not a lot of choice about what to do next if the navigation strategy does not halt the search. The choice of the next vertex to move to is made by the navigation strategy, so is the number of elements of $N_\beta(v)$ to examine before making the move. Having examined a number of next possibilities, the navigation strategy selects one, usually on the basis of the fitnesses observed.

The particular solutions to the choices above are usually responsible for the name given to the algorithm. For example, if all the neighbors are examined and one of those with maximal fitness is chosen to move to, the hillclimbing algorithm is called *Steepest Ascent*. If the navigation strategy examines neighbors until one with better fitness is found and then moves to it, we shall call the hillclimbing algorithm *Any Ascent*. Other aspects of algorithms that will be considered part of the navigation strategy include such things as: Deciding when to stop, deciding which operators to use and when, deciding how and when to change the algorithm's temperature variable (should one exist), deciding to adjust the representation or the mapping Γ between the object space ($\mathcal{O}$) and representation space ($\mathcal{R}$) to focus on a specific area of $\mathcal{O}$, deciding on population size (if the algorithm maintains a population), and deciding on a balance between exploration and exploitation.

3.3. Crossover Landscapes

In order to understand the workings of the crossover hillclimbing algorithm, it is necessary to have an idea of the structure of crossover landscapes. As was briefly mentioned in Chapter 2, crossover landscapes are not simple connected graphs. This dissertation does not attempt a full study of the structural properties of crossover landscapes, however, several crossover landscapes are shown in Figures 8 to 11. In each of these, $\mathcal{R}$ is the space of binary strings of length two or three. Figure 8 is identical to Figure 4 on page 35 and is reproduced here so as to have these figures close to each other for easy comparison. It shows the landscape generated by the one-point crossover operator that produces two children from two parents ($\chi_1^{2\rightarrow 2}$). Figure 9 shows the landscape generated by two-point crossover that produces two children from two parents ($\chi_2^{2\rightarrow 2}$), Figure 10 shows the landscape generated by uniform crossover that produces two children from two parents ($\chi_u^{2\rightarrow 2}$), and Figure 11 is the landscape generated by one-point crossover that produces one child from two parents ($\chi_1^{2\rightarrow 1}$). A more general figure for parameterized uniform crossover [95] would have probabilities on the edges, similar to those in Figure 3 on page 33.

3.4. Experiment Overview

The crossover hillclimbing algorithm was constructed to study the performance of crossover in relative isolation. One problem with assessing the importance of the operator is that it is difficult to dissociate the operator and its effects from the rest of the machinery of the GA. Naturally, the power of the operator, if any, might arise from its actions in concert with other operators and this needs to be kept in mind. However, the crossover hillclimber allows a reasonably simple view of crossover and the results obtained with the algorithm throw light on the GA itself.

The CH algorithm is compared on a number of functions to three algorithms: Two simple GAs (one with and one without elitism) and a simple mutational hillclimber. This collection of algorithms is interesting as the two hillclimbing algorithms use a single (different) operator and these two operators are both employed by the GAs. Otherwise, the two hillclimbers have very similar navigation strategies. The GAs of course have a much more sophisticated navigation strategy. The algorithms are compared exclusively on the basis of the expected number of evaluations they require to reach certain levels of performance.[1] It should be emphasized that the purpose of this experiment was *not* to do an exhaustive comparison of the four algorithms. This would be an extremely large undertaking as there are many variants of the GAs and many parameters that could be adjusted. Instead, a fairly standard collection of parameter settings for these algorithms was chosen and held fixed across all experiments.

On several occasions I performed some initial testing to see whether a particular parameter setting appeared to increase an algorithm's overall performance. In each case I chose the option that resulted in the higher performance. Virtually all this parameter testing was for the GAs, though it was necessary to change the mutational

[1] This is a practical yardstick, and is motivated by the belief that when choosing an algorithm to solve a problem, one is chiefly motivated by the amount of time that will be spent awaiting a solution. If the number of function evaluations performed by an algorithm is proportional to its run time, then concentrating on the expected number of evaluations is appropriate.

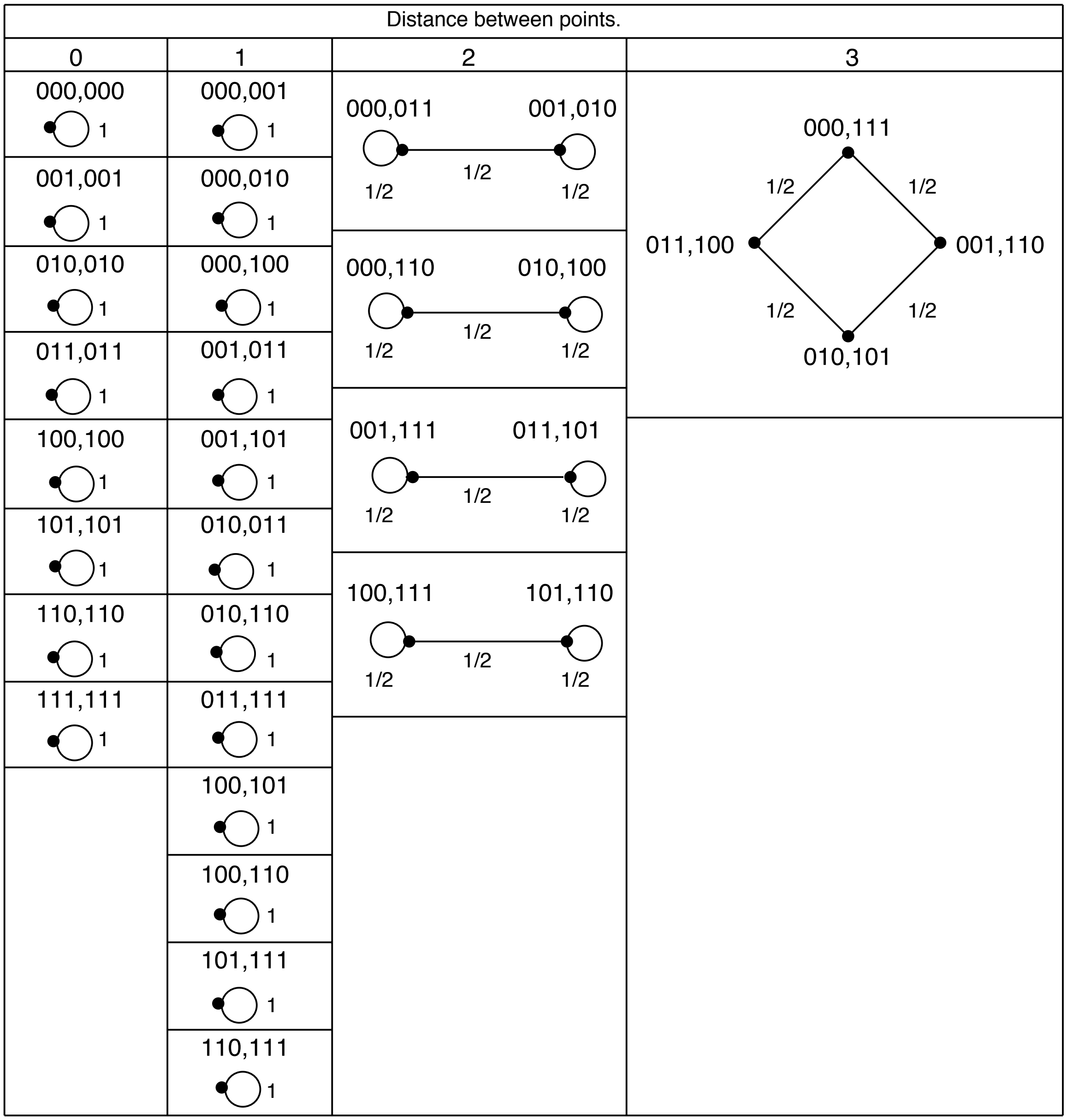


Figure 8. The one-point crossover landscape for binary strings of length three. The crossover operator, $\chi_1^{2\to 2}$, produces two offspring from two parents. Edges are bidirectional.

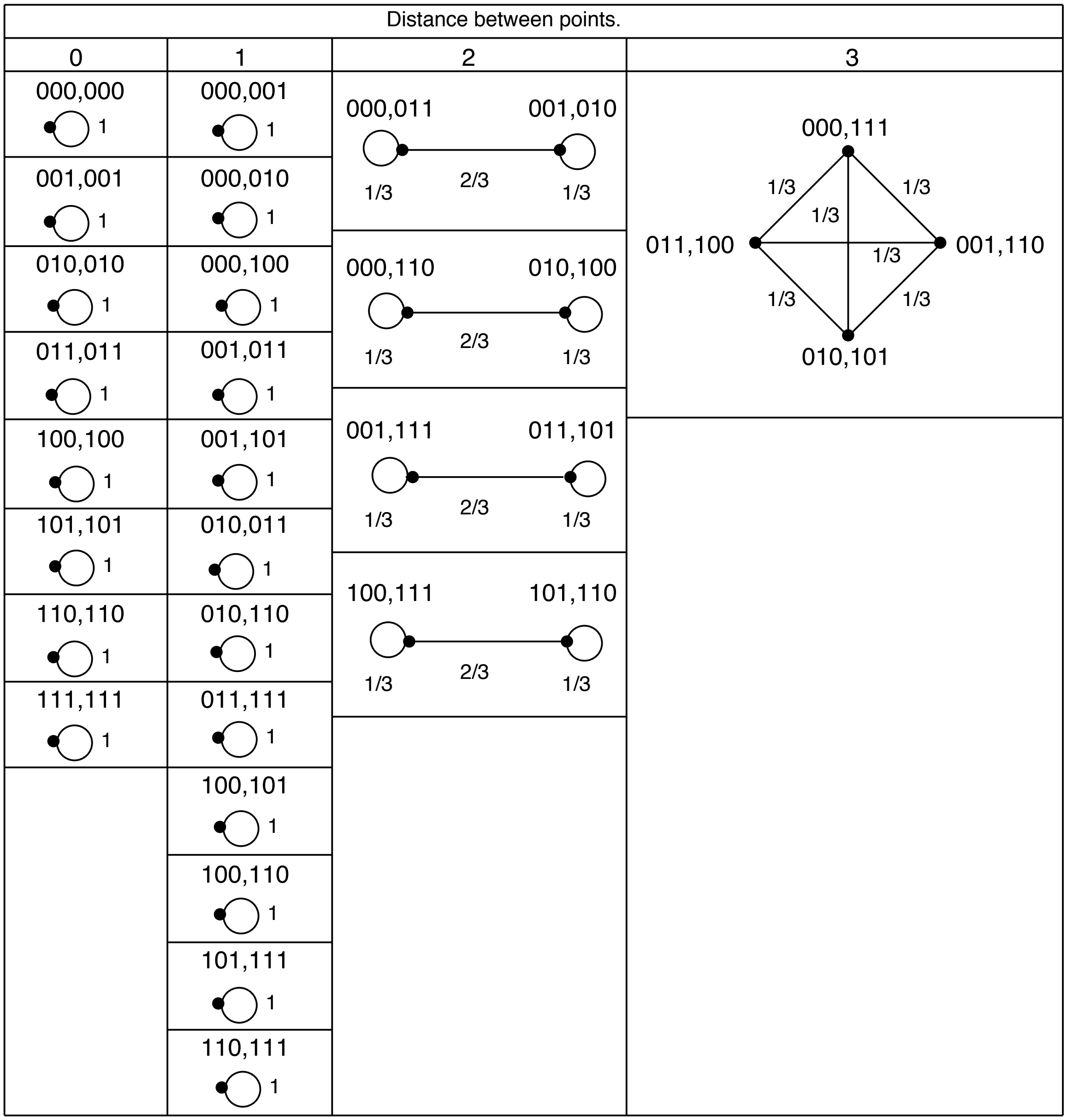


Figure 9. The two-point crossover landscape for binary strings of length three. The crossover operator, $\chi_2^{2\to 2}$, produces two offspring from two parents. Edges are bidirectional.

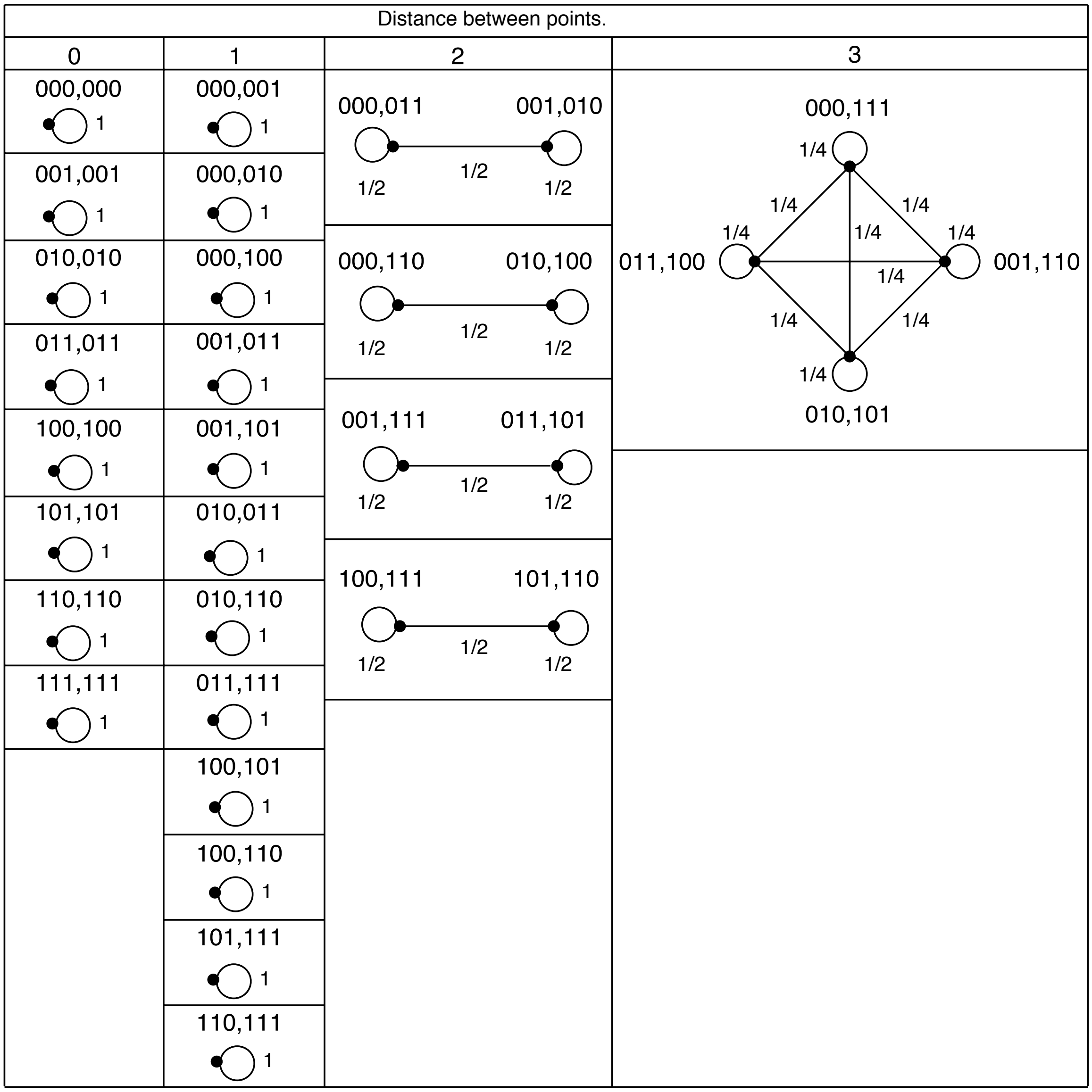


Figure 10. The uniform crossover landscape for binary strings of length three. Bits are chosen from parents with probability 1/2. The crossover operator, $\chi_u^{2\rightarrow 2}$, produces two offspring from two parents. Edges are bidirectional.

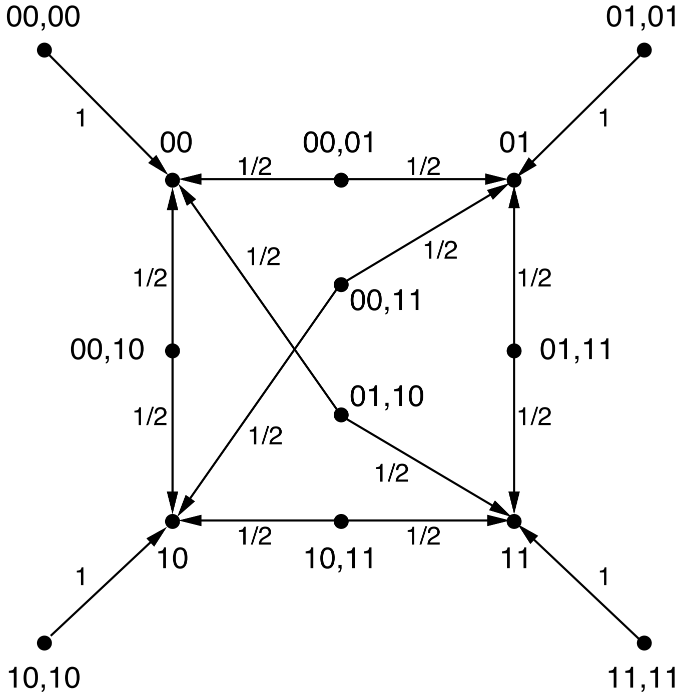


Figure 11. The one-point crossover landscape for binary strings of length two. The crossover operator, $\chi_1^{2\to1}$, produces one offspring from two parents (and is consequently not walkable). This is done by generating both offspring (as in $\chi_1^{2\to2}$) and then selecting one uniformly at random to retain. Other offspring selection methods would alter edge probabilities.

hillclimbing *algorithm* (from the description given by Forrest et al. [96]) to increase its performance. This is described in §3.6(55). The crossover hillclimber was surprising in its robustness. The parameter settings used were chosen after a few very preliminary experiments and not altered until the reasons for the success of the algorithm were later sought.

3.5. The Simple Genetic Algorithm

The following parameter settings were used in the GAs. These can all be defended as fairly standard. Population size was 500 (initial experiments with smaller sizes resulted in a marked decrease in the GA's performance). The maximum number of generations was 100. This many generations were run unless the population became too converged or a point with maximal fitness was encountered. The maximum amount of convergence permitted in a population was 0·85 at every locus. If every locus in the population converges to at least this frequency, the run is terminated. The per-locus mutation probability was 0·01. The probability that crossover was applied to a pair before they were added to the next generation was 0·75. To avoid complications with selection, binary tournament selection [97] was used as it is rank-based, avoiding any need to scale population fitness or otherwise consider the fitness function, even across experiments. The winner of the tournament was given a 0·75 chance of being selected. See [98, 99] for discussion on the advantages of rank-based selection schemes.

A major omission from this list of parameters is some form of elitism [30], either an explicit copying of the best individual or a generation gap less than one to decrease the probability of discarding the best structure found so far. There has been recent discussion on the use of GAs as function optimizers [100, 101], perhaps stemming from Holland's reminder that the original intent of the GA was more one of improvement than one of optimization [102]. It is not clear whether a standard GA should include some form of elitism or not [5, 9, 30, 103]. Grefenstette's standard GA [103], which takes its lead from De Jong [30] recommends some form of elitism. De Jong refers to a "canonical" GA (which does not use elitism) and mentions elitism as a method of improving GA performance when a GA is used as a function optimizer [100].

It is perhaps unfair to compare the GA against two algorithms that use a very elitist hillclimbing strategy, but then again, they *are* different algorithms. The obvious solution is to examine both possibilities. A summary of the result of this is that elitism never resulted in worse performance (usually the GA with elitism would attain difficult performance levels in five to ten times fewer function evaluations than the GA without elitism), but that the change never resulted in a significant change in the positional ranking of the GA with respect to the other two algorithms. For example, if the performance on a problem of a GA without elitism was between the other algorithms, then the GA with elitism would perform better than the GA without elitism, but still between the other algorithms. In what follows, the standard and elitist versions of the GA are denoted by GA-S and GA-E. Abbreviations for all the algorithms of this chapter can be found in Table 1.

3.6. The Bit-flipping Hillclimbing Algorithm

The bit-flipping hillclimbing algorithm (BH) is essentially identical to the "random mutation hillclimbing" (RMHC) algorithm suggested to Forrest et al. [96] by Richard Palmer. The algorithm chooses a random starting vertex and thereafter proceeds to any β-neighbor with equal or higher fitness as soon as one can be located. On the

Table 1. Algorithm abbreviations and brief descriptions. The last column gives the page on which a fuller description of each algorithm can be found.

Abbreviation	Description	Page
GA-S	Standard GA.	54
GA-E	Elitist GA.	54
BH	Random Mutation Hillclimbing.	55
CH(a,s,j)	Crossover Hillclimbing, a attempts, s steps, j jumps.	57
CH	CH(10,3,1000).	57
CH-1S	CH(10,1,3000) One step per hypercube.	74
CH-NJ	CH(10,3000,0) No jumps between hypercubes.	74
GA-RC	GA with random crossover.	78
GA-RCE	Elitist GA with random crossover.	78
BH-MM	Hillclimbing with macromutations.	81
BH-DMM	Hillclimbing with distributed macromutations.	81

royal road functions examined by Mitchell, Forrest and Holland [36, 96], the algorithm has the important property of moving indefinitely on a β-plateau. The royal road functions have many such sets, which are not β-mesas and so are eventually escaped by this algorithm. This accounts for its success on these problems over other hillclimbers that use β but do not make moves to points of equal fitness.

In general, the operators and representation chosen to attack a search problem may create many plateaus that *are* mesas. When BH encounters a β-mesa it will wander on it indefinitely. For this reason, the algorithm has a parameter that limits the number of steps that may be taken without a fitness increase. This parameter is the only difference between BH and RMHC. If this number (which was set to 10,000) is exceeded, the search is terminated. Without such a limit, BH encounters chronic problems on the busy beaver problem (described below) as a result of the choice of representation for that problem. The maximum number of random mutations to try before deciding that a point has no equal or higher neighbor and terminating the search was set to twice the number of neighbors.

3.7. The Crossover Hillclimbing Algorithm

The CH algorithm has a navigation strategy that is very similar to that of the BH algorithm just described. However, the operator used here is two-point crossover (in which two offspring are produced from two children).

The two-point crossover landscape on binary strings described in §3.3(49) is a family of hypercubes, each of whose vertices corresponds to a pair of individuals from $\mathcal{R}$. To conduct a hillclimb within a hypercube, randomly choose a starting pair (A_0, B_0) and perform crossover on them repeatedly until some limit (*max-attempts*) is reached or a pair (A_1, B_1) is found with $\max(f(A_1), f(B_1)) \geq \max(f(A_0), f(B_0))$. If no such pair is found, halt the hillclimb. Otherwise, continue, performing a similar search from (A_1, B_1). This process carries out a single climb within a hypercube of the landscape.[2] Once *max-attempts* crossovers have been tried and a new vertex containing a better individual has not been found, there is no need to discard both A_i and B_i to continue the search. Instead, the CH algorithm discards the less fit of A_i and B_i (ties are broken arbitrarily) and replaces it with a randomly chosen element of $\mathcal{R}$. This has the effect of (probably) transferring the search into another hypercube, from which the hillclimb may continue in identical fashion. Figure 12 shows a fragment of such a climb. A second parameter, *max-jumps* limits the number of times the jumping between hypercubes may occur. A third, *max-steps* can be used to limit the number of steps that may be taken consecutively within a hypercube before a jump is forced. This is equivalent to saying that the number of pairs in the sequence of steps $(A_0, B_0), (A_1, B_1), \cdots$ within a hypercube is limited to $max - steps + 1$. A triple representing the value of these three parameters, *max-attempts*, *max-steps* and *max-jumps* can be used to describe a particular CH algorithm. For example, CH(10,3,1000) represents the CH algorithm that examines up to 10 neighbors (under crossover), limits the number of steps that may be consecutively taken within a hypercube to three, and makes at most 1000 jumps via discarding the worst individual of the current pair when no acceptable crossover is found or when three steps have been taken within a hypercube. In the initial experiments of this chapter, we will be concerned only

[2] Naturally, there is a chance (typically high) that the global optimum (assuming there is only one) cannot be discovered from a randomly chosen starting pair. With binary strings of length l, this probability is $1 - (3/4)^l$.

with CH(10,3,1000) and will use CH to mean this algorithm.

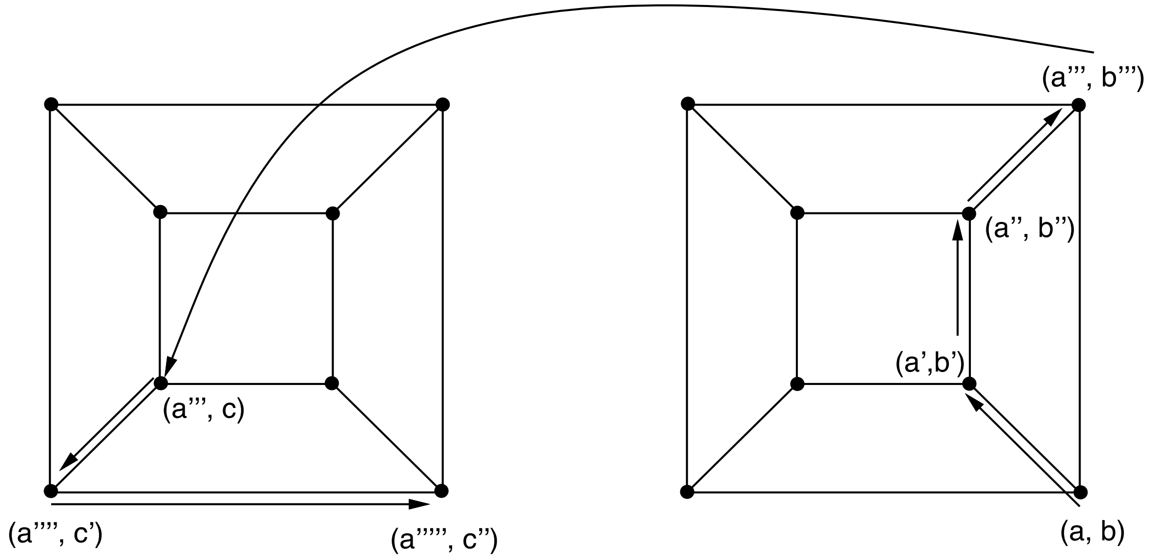


Figure 12. A fragment of a hillclimb under crossover. Three uphill steps are taken (using crossover) within the right hypercube, from (a, b) to (a''', b'''). At that point b''' is discarded and replaced with the randomly generated c. Two more steps are then taken on the left hypercube. The jump into the left hypercube is either the result of a limit on the number of steps in a hypercube (max-steps) or a limit on the number of attempts to find an improving crossover (max-attempts).

3.8. Test Problems

This section describes the set of six problems that were studied. Each of these problems has a fixed length string representation. Three instances of each of the six problems were studied, resulting in eighteen separate problems in six classes.

3.8.1. One Max

This problem was studied using a GA (and other algorithms) by Ackley in [26]. The task is simply to locate a binary string which consists of as many ones as possible. The fitness of a string is the number of ones it contains (Ackley used ten times this). Problems in which the fitness of an individual is a function of the number of ones in the string have been termed *functions of unitation* by Goldberg and have received a considerable amount of attention, particularly in the study of deception [104]. While the problem clearly has a single global maximum and no β-local-optima, there are local optima for other operators, as demonstrated by Culberson [29], who shows that the problem contains $\chi_1^{2\to 2}$-local-optima. The three instances of the problem that were studied had lengths 30, 60 and 120.

3.8.2. Fully Easy

This problem requires the solution of a number of fully easy subproblems. Each subproblem is a six bit function of unitation. The function values for the different unitation values are arranged in such a way that all low-order schema effectively point the way to the global optimum. This function was designed by Deb and Goldberg [105]. The function values according to unitation (number of ones present in the string) are shown in Table 2.

Table 2. A fully easy 6-bit problem. Unitation is the number of bits in the binary string that are set to one. The maximum fitness point is the string 000000 with zero ones.

Unitation	0	1	2	3	4	5	6
Fitness	1·0	0·8	0·6	0·9	0·5	0·7	0·9

Fully easy problems were designed with GAs in mind, and their name reflects reasoning that they should be easy for GAs to solve. This does not imply that they will be easy for other forms of search—by observation, the problem contains β-local-maxima. For instance, a steepest ascent hillclimber using β would reach a local maximum if it started to climb from any string with more than a single one bit. The global optimum, the string with six zeroes, has a basin of attraction that includes only seven of the sixty-four possible strings. Each six bit problem has $\binom{6}{3} + \binom{6}{6} = 21$ β-local-maxima, which is almost a third of the entire space. A discussion of the problem's difficulty for BH can be found below. Each problem instance was constructed by concatenating some number of these 6-bit problems together. The three experiments used 5, 10 and 15 such subproblems, resulting in problem sizes of 30, 60 and 90 bits.

3.8.3. Fully Deceptive

The fully deceptive problems are also representative of a class of problems that have received wide attention in the study of GAs. The particular problem addressed here is from the same source as the fully easy problem, [105], and the problem instances also have 5, 10 and 15 subproblems each of six bits. These problems get their name from reasoning that indicates that they should be difficult for GAs. The low-order schema all lead the GA away from the single global maximum. The subproblem appears in Table 3.

Table 3. A fully deceptive 6-bit problem. Unitation is the number of bits in the binary string that are set to one. The maximum fitness point is the string 111111 with six ones.

Unitation	0	1	2	3	4	5	6
Fitness	0·90	0·45	0·35	0·30	0·30	0·25	1·00

The first thing to notice about this problem is that the location of the global maximum is the string of all ones, the opposite of the fully easy problem. This problem has one global maximum (all ones) and one β-local-maximum (all zeroes). Under steepest ascent, the global maximum can be reached from only seven points in the space. The β-local-maximum will be reached from the rest of the space.

3.8.4. Distributed Fully Deceptive

The distributed deceptive problem contains a number of fully deceptive subproblems, identical to those in the fully deceptive problem above. The difference is that the bits comprising each subproblem are spread through the string. Thus if the problem contains five 6 bit-subproblems, the bits for the first subproblem are located at positions 0, 5, 10, 15, 20 and 25. The bits for the second subproblem are located at positions 1, 6, 11, 16, 21 and 26 and so on.

3.8.5. Busy Beavers

In Rado's "busy beaver" problem [106], deterministic Turing Machines (TMs) with alphabet {0, 1} and a fixed number of states k (plus a halt state) are sought that write as many 1 symbols as possible on an initially zero-filled two-way infinite tape before halting. The number of 1 symbols that remain on the tape is the machine's score. The problem is to find, for a given k, a halting TM with the highest score, which will be denoted by $\Sigma(k)$. More than one machine may generate $\Sigma(k)$ ones. The busy beaver problem is extremely difficult and has an interesting history. The problem is solved for $k \leq 4$. There are two 5-state machines that halt and leave 4,098 ones on the tape and a 6-state machine that produces over 95 million! See Brady [107, 108] for details of $k = 4$ and Marxen and Buntrock [109] for $k = 5$. The results of applying a GA to the problem and the original presentation of reverse hillclimbing (the subject of Chapter 4) can be found in [110].

3.8.5.1. Fitness and Representation

The fitness of a TM was its score, or -1 if the machine did not halt. The general halting problem is not an issue for $k \leq 4$ (as in our experiments), since the problem is solved for machines with up to 4 states. In each case, the maximum number of steps taken by a halting machine is known and can be used to terminate evaluation.

A TM with k states was represented by a character string of $6k$ bytes. When the machine is in a certain state, looking at a certain symbol, it needs to know three things: the next state, the symbol to write on the tape and the direction in which to move. Since the machine could be scanning either a 0 or a 1, six bytes are required. By restricting crossover to byte boundaries, only legal TM's were generated.

This representation allows the use of simple operators. There are a number of symmetries that other approaches have taken advantage of that are not exploited here. For example, the direction the tape head moves when the machine enters the halt state makes no difference to the number of ones left on the tape. Noticing this, and including it in the representation halves the size of the search space. Similarly, the symbol written to the tape on the transition into the halt state can be assumed to be a 1 as this assumption cannot reduce the number of ones the machine has written. Other symmetries include the renaming of states, and replacing left moves with right and vice-versa.

The "representation" that has been most widely used in other work on this problem is known as Tree Normal Form. This takes advantage of all the above symmetries. Unfortunately it is not so much a representation as an algorithm for producing and running TMs. If such a representation were adopted, and there is no doubt that it could be, it would also require the construction of specialized operators designed to produce TMs of the correct form. These issues are exactly those that were discussed in §1.3(5). Once again, intelligent observations about the nature of the problem lead to representations that greatly reduce the size of the problem. For example, for 4 states TNF produces approximately 600,000 machines instead of 25,600,000,000.

3.8.5.2. Mesas

The combination of representation and operator here results in a landscape with many mesas. The operator used by BH changes any state into any other, a left movement into a right or a one into a zero. Every halting TM in the landscape is connected by an edge to a twin that produces the same number of ones. This corresponds to changing the direction of movement when entering the halt state. Thus every TM with no fitter neighbors is part of a mesa of size at least 2 which, unless modified, BH can never escape. This was the motivation for imposing a limit on the number of non-improving steps that BH allows.

It is also interesting to note that every halting TM is connected by an edge to at least one TM that does not halt. A halting TM can be made to not halt by modifying the behavior when in the initial state so it remains in the state (either symbol may be written and either direction may be taken). Thus every machine with maximal fitness also has at least one neighbor that has lowest fitness. Similarly, each TM with a single transition into the halt state can be made to not halt by altering this transition. These phenomena are the result of a simplistic choice of representation and operators.

3.8.6. Holland's Royal Road

The Royal Road functions were introduced by Mitchell, Forrest and Holland [36]. They were designed as functions that would be simple for a genetic algorithm to optimize, but difficult for a hillclimber. However, a form of hillclimbing suggested by Palmer [111], easily outperformed a genetic algorithm on what had been expected to be the simplest Royal Road function for that algorithm. Recently, Holland [112] presented a revised class of Royal Road functions that were designed to create insurmountable difficulties for a wider class of hillclimbers, and yet still be admissible to optimization by a genetic algorithm. There is no easily understandable presentation of these functions available elsewhere. This section provides one and is somewhat lengthy as a result.

Holland used seven variables in the description he presented to the genetic algorithms community. In describing the class of functions, I will use his variable names where reasonable, and indicate deviations. Holland suggested a default value for each of these variables and in what follows I will use these values to provide illustration. The following terms will be used consistently in what follows: *block*, *complete*, *gap*, and *region*. These are introduced below and can be relied on to always mean exactly the same thing.

3.8.6.1. Description

The function takes a binary string as input and produces a real value which the searcher must maximize. The string is composed of 2^k non-overlapping contiguous regions, each of length $b+g$. With Holland's defaults, $k=4$, $b=8$, $g=7$, there are 16 regions of length 15, giving an overall string length of 240. We will number the regions, from the left, as $0, 1, \ldots, 2^k - 1$.

Each region is divided into two non-overlapping pieces. The first, of length b, will be called the block. The second, of length g, will be called the gap.[3] In the fitness calculation, only the bits in the block part of each region are considered. The bits in the gap part of each region are completely ignored during fitness calculation. Holland's description called the block part of each region by various names: "building blocks," "elementary building blocks," "schemata," "lower level target schemata," and "elementary (lowest-level) building blocks (schemata)."

The fitness calculation proceeds in two steps. Firstly, there is what Holland calls the PART calculation. Then follows the BONUS calculation. The overall fitness assigned to the string is the sum of these two calculations.

The PART Calculation

The PART calculation considers each block individually, and each in exactly the same fashion. Each block receives some fitness score, and these are added to produce the total PART contribution to the overall fitness. The fitness of each block is based entirely on the number of 1 bits it contains.[4] The idea is to reward 1's up to a certain point. Every 1 up to (and including) a limit of m^* adds to the block's fitness by v. Thus, with the default settings, a block with three 1's would have fitness $3 \times 0{\cdot}02 = 0{\cdot}06$. If a block contains more than m^* 1's, but less than b 1's, it receives $-v$ for each 1 over the limit. With the default settings $m^* = 4$, and so a block with six 1's is assigned a fitness of $(6-4) \times -0{\cdot}02 = -0{\cdot}04$.

[3] Holland did not give a name for this part of the regions, or give a variable name for its length. These gaps have been called "introns" (borrowing from biology) and have been used with varying degrees of success [96, 113] to alter the effects of crossover in genetic algorithms.

[4] Holland used $m(i)$ to denote the number of 1's in block i. I will not use a variable.

Finally, if a block consists entirely of 1's (i.e., it has b 1's), it receives *nothing* from the PART calculation. Such a block will be rewarded in the BONUS calculation. If a block consists entirely of 1 bits, it will be said to be *complete*. From the above, we can construct a table of fitness values based on the number of ones in a block. Table 4 gives the values for the default settings.

Table 4. PART fitness values for Holland's default settings. Unitation is the number of bits in the binary string that are set to one. Although the $\binom{8}{4}$ strings with unitation 4 have the highest PART fitness, these strings are not optimal in the complete function as the BONUS calculation assigns a fitness of 1·0 to the string 11111111.

Unitation	0	1	2	3	4	5	6	7	8
Block fitness	0·00	0·02	0·04	0·06	0·08	−0·02	−0·04	−0·06	0·00

The BONUS Calculation

The idea of the BONUS calculation is to reward completed blocks and some combinations of completed blocks. Holland gives rewards for attaining what he calls "levels." At the lowest level, zero, rewards are given for complete blocks (i.e., blocks that consist entirely of 1's).[5] If such a block exists, it receives u^* as its fitness. Any additional complete blocks receive a fitness of u. Thus, with the default settings, a string that contained three complete blocks would receive a BONUS fitness of $1{\cdot}0 + (2 \times 0{\cdot}3) = 1{\cdot}6$ for level 0.

But that is not the end of the BONUS story. At the next level, *pairs* of blocks are rewarded. Suppose we label the blocks from left to right as $B_0, B_1, \ldots, B_{2^k-1}$. The function then rewards any completed pair (B_{2i}, B_{2i+1}) for $0 \leq i < 2^{k-1}$. The rewards are calculated in the same way as they were done at level 0. The first completed pair of blocks receives u^*, and additional completed pairs receive u. Notice that not all pairs of completed blocks are so rewarded. Only those contiguous pairs whose indices are of the form $(2i, 2i+1)$.

We can now calculate rewards for the next level, that of quadruplets of completed blocks, and this is done in the same fashion. In general, there are $k+1$ levels, and at level $0 \leq l \leq k$, we are looking for contiguous sets of 2^l complete blocks, whose first

[5] Holland numbered his lowest level 1 not 0.

block is labeled B_{i2^l} with $0 \leq i < 2^{k-l}$. At all levels, the first such set of complete blocks receives fitness u^*, and additional sets of completed blocks receive fitness u. The total fitness for the level is the sum of these fitnesses.

To make this more concrete, consider the function with Holland's default values. With $k = 4$ we have 16 regions and each contains a block of length $b = 8$. The BONUS fitness calculation rewards completed single blocks (this is level 0), and rewards the completion of the sets $\{B_0, B_1\}$, $\{B_2, B_3\}$, $\{B_4, B_5\}$, $\{B_6, B_7\}$, $\{B_8, B_9\}$, $\{B_{10}, B_{11}\}$, $\{B_{12}, B_{13}\}$, $\{B_{14}, B_{15}\}$, (level 1) $\{B_0, \ldots, B_3\}$, $\{B_4, \ldots, B_7\}$, $\{B_8, \ldots, B_{11}\}$, $\{B_{12}, \ldots, B_{15}\}$ (level 2) $\{B_0, \ldots, B_7\}$, $\{B_8, \ldots, B_{15}\}$ (level 3) and finally $\{B_0, \ldots, B_{15}\}$ (level 4) of completed blocks. The total BONUS contribution to the fitness is computed by adding the fitness at each of the $k + 1$ levels.

3.8.6.2. Experiments

The experiments on Holland's royal road in this chapter only varied k, which determines the number of level zero blocks and hence the number of levels. The three values used were 2, 4 and 6, giving problems with 60, 240 and 960 bits.

3.9. Results

The results of twelve of the eighteen experiments are shown in Figures 13 to 24. Results for the smallest instance of each problem are not shown. Performance on the smallest instances of each problem qualitatively matched performance on the larger instances. In all these graphs, the X axis (mean number of evaluations) has a log scale. Each of the algorithms was run at least 2000 times on each instance of each problem and in most cases at least 10,000. Each time an algorithm achieves a performance level (for instance, completing a fully deceptive subproblem), the number of evaluations is recorded. From this, the mean number of evaluations taken to achieve performance levels is plotted for each algorithm on each problem. The line representing an algorithm is terminated at the highest level that was achieved at least ten times. Standard errors for the following graphs are presented in Tables 40 to 69 in Appendix C.

3.9.1. Overall Trends

A quick perusal of the result graphs shows that the CH algorithm has performed remarkably well. On these experiments it is certainly the most consistent algorithm—in no experiment does it have the worst performance. The hillclimber is clearly the worst algorithm on four of the functions (fully deceptive, distributed fully deceptive, fully easy and Holland's royal road), and the two GA variants are clearly the worst algorithms on the remaining two (busy beavers and one max). On three of the functions (fully deceptive, fully easy and Holland's royal road), the CH algorithm is the best performer and it can easily be argued that it is the best on a fourth (busy beavers). Only on one problem of the six, one max, is it significantly beaten (by BH, which is hardly surprising).

It is interesting to note how much consistency there is in the graphs for each experiment. Each of the six experiments has three graphs and within each set of three there is remarkable consistency. The shapes of the lines representing each algorithm are well preserved over the graphs, as are the relative orderings of the algorithms and the ways that the orderings change, and when they change, as higher performance levels are achieved.

The addition of elitism to the GA resulted in improvement in every case (though some were very minimal). However, in no case was the improvement great enough to significantly alter the relative ranking of the GA without elitism. For example, if the order of the three algorithms CH, GA-S and BH was, say, BH, GA-S, CH from best to worst, then the performance of GA-E would fall between that of BH and GA-S.

Table 5 gives a rough overview of the performance of the algorithms. Each algorithm is ranked for each problem, with a rank of 1 being the best. When it was

Table 5. Overall trends in the crossover hillclimbing experiment. Algorithms are roughly ranked according to performance on the six problems. 1 = best, 3 = worst. The standard and elitist versions of the GA are represented as a single column.

	BH	CH	GA
Busy beavers	1	2	3
One max	1	2	3
Fully deceptive	3	1	2
Distributed deceptive	3	1	1
Fully easy	3	1	2
Holland's royal road	3	1	2

not clear which algorithm was better from looking at the three experiments for a problem, preference was given to the algorithm that performed best on the largest instance of the problem and algorithms received equal rank if there was still no clear difference. The two versions of the GA have been coalesced since they virtually always rank next to each other. This allows a clearer view of the overall pattern. It should be clear that CH is never the worst, the GA is never clearly the best and BH is either best or worst.

3.9.2. One Max

Figures 13 and 14 show the results of the four algorithms on the one max problem described in §3.8.1(58). These are consistent with what we might expect from previous work by Ackley [26] and Culberson [29]. Ackley's results show two versions of a hillclimber clearly outperforming two versions of a GA. Culberson demonstrated the existence of $\mathcal{X}_1^{2\to 2}$-local-maxima in the component of the one-point crossover landscape that corresponds to complementary binary strings for the one max function. There are $\mathcal{X}_2^{2\to 2}$-mesas in components of the two-point crossover landscape for non-complementary strings. A trivial example is any pair (a, a) where a is not the global maximum. In general, if $f(a,b) = \max\{g(a), g(b)\}$ and $g(v)$ is the number of ones

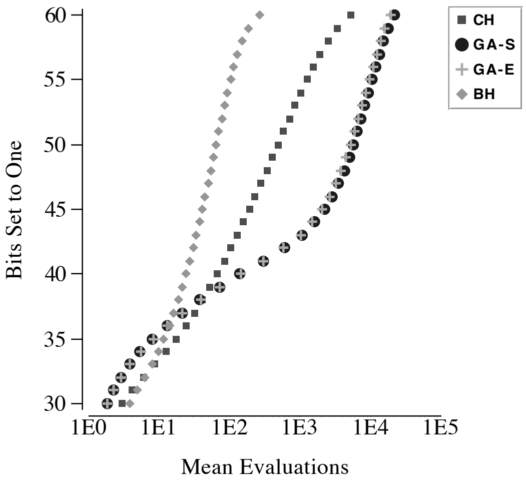


Figure 13. Mean evaluations to solve a 60-bit one max problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

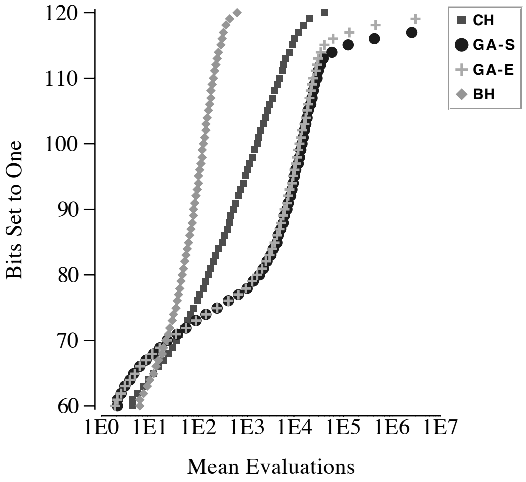


Figure 14. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

in the string v, then (a, b) will be a $\mathcal{X}$-local-maximum if b has a 0 everywhere a does and a is not the global maximum (or vice-versa). For example, $(1101101, 1001001)$ is a $\mathcal{X}$-local-maximum since no crossover (of any standard type) can produce a pair containing a member with more than five 1s.

Thus the crossover landscape contains $\mathcal{X}_2^{2\to2}$-mesas and $\mathcal{X}_2^{2\to2}$-local-optima, and these will naturally degrade the performance of a simplistic hillclimbing navigation strategy that uses $\mathcal{X}_2^{2\to2}$. By choosing a new partner for the best string of the current pair, these can be escaped, but this is only performed a fixed number of times. As the algorithm comes closer to locating the global maximum, the probability of jumping to a $\mathcal{X}_2^{2\to2}$-local-maximum by choosing a new partner increases. Of course, a vertex may not be a $\mathcal{X}_2^{2\to2}$-local-maximum but because the navigation strategy of CH only examines a small number of neighbors (10), it may not find the crossover that would produce the fitness increase.

The BH algorithm has a very similar navigation strategy but the landscape it is operating on has no β-local-maxima and so it can never fail to find the global maxima if it is run sufficiently long. Also, this algorithm examines as many as twice the number of neighbors while looking for an uphill direction, so it is unlikely to miss an opportunity to increase fitness when this is possible.

3.9.3. Fully Easy

Figures 15 and 16 show the results of the four algorithms on the fully easy problem described in §3.8.2(59). The most striking feature of these graphs is the vastly superior performance of the CH algorithm on the largest of these problems (15 fully easy subproblems of length 6). Not only does the CH algorithm maximize all 15 blocks, it does so using less evaluations on average than GA-S uses to maximize 11 of them (the highest level GA-S achieves). The CH algorithm achieves that level (11 blocks), using an average of almost 1000 times fewer evaluations than GA-S. GA-E does better, but CH is still using two orders of magnitude fewer evaluations at the highest level achieved by GA-E.

The performance of BH on this function is particularly poor. As discussed in §3.8.2(59), each subproblem can only be solved by steepest ascent hillclimbing (using β) from 7 of the 64 possible starting points. The situation is not much better for the BH algorithm. In addition to those seven starting points (from which it will find the maximum with probability 1 if left to run long enough), there are $\binom{6}{2} = 15$ strings (with 2 ones) from which the BH will find the global maximum with probability one-third. Thus BH can be expected to solve each fully easy subproblem with a

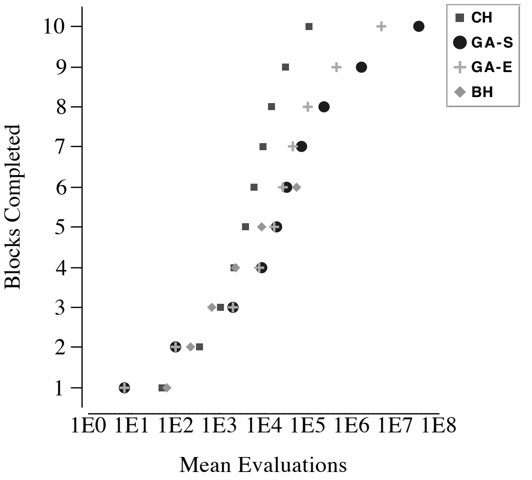


Figure 15. Mean evaluations to solve 10 fully easy 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

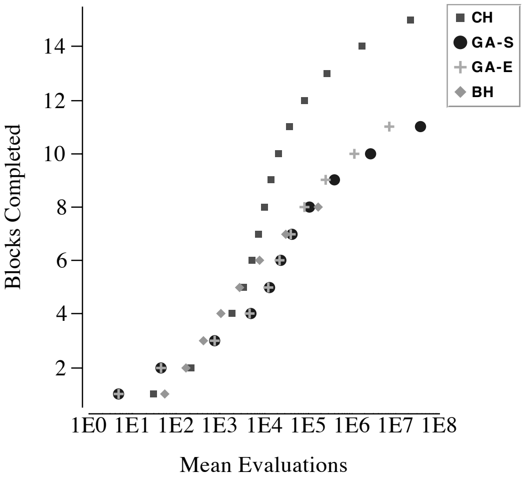


Figure 16. Mean evaluations to solve 15 fully easy-6 bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

probability of only 12/64.

Even if the landscapes generated by mutation and crossover were equal in terms of numbers of peaks and the sizes of their basins of attractions (from the point of view of the hillclimbing algorithms involved), CH would have an advantage over BH. This stems from the navigation strategy it employs. The CH algorithm operates on a pair of points, and thus it is not necessary to consider a hillclimb finished just because the current pair is not making progress. The worse of the two is discarded and replaced with a random new partner which may take the search to another component of the crossover landscape and perhaps away from a $\mathcal{X}$-local-maximum. It would perhaps be fairer to allow the BH algorithm to do approximately the same thing and randomly set some fraction of its bits when what appears to be a β-peak is reached. This algorithm was used by Palmer and Pond, who found that replacing approximately a quarter of the bits in the current individual worked best on their problems [114].

3.9.4. Fully Deceptive

Figures 17 and 18 show the results of the four algorithms on the fully deceptive problem described in §3.8.3(60). The results are very similar to the results for the fully easy problem of §3.9.3(68), though with less disparity between the CH algorithm and the GA. Once again, CH performs very well, requiring approximately ten times fewer

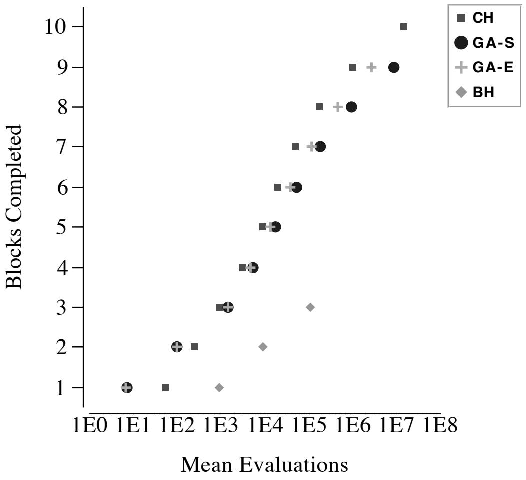


Figure 17. Mean evaluations to solve 10 fully deceptive 6-bit problems with crossover hill-climbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hill-climbing (BH).

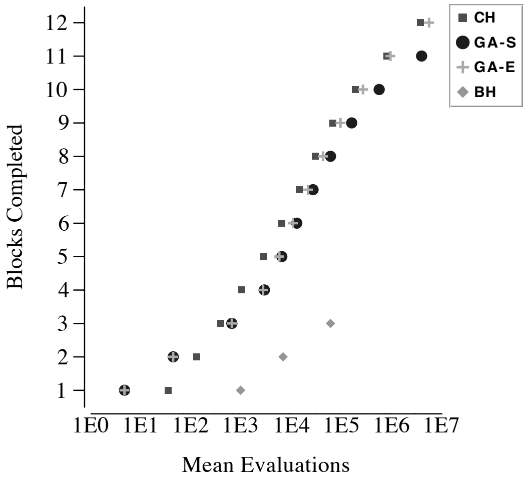


Figure 18. Mean evaluations to solve 15 fully deceptive 6-bit problems with crossover hill-climbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hill-climbing (BH).

evaluations than GA-S to achieve the highest performance levels. The performance of GA-E was quite close to that of the CH algorithm on the largest of the problems. No algorithm managed to optimize all 15 deceptive blocks in the hardest experiment. BH performed very poorly.

3.9.5. Distributed Fully Deceptive

Figures 19 and 20 show the results of the four algorithms on the distributed fully deceptive problem described in §3.8.4(60). CH and the GA-E perform the best on this function. Because their performances are so close, they are both given a 1 ranking in Table 5 on page 66. It is interesting to note that the crossover-based algorithms do better than the bit-flipping based BH, despite the representation being very poor for crossover. The performance of BH is virtually identical to that of the fully deceptive problems, which is as it should be since the distributed representation will not affect this algorithm.

3.9.6. Busy Beavers

Figures 21 and 22 show the results of the four algorithms on the busy beaver problem described in §3.8.5(60). In previous work (with G. Rawlins) I compared the

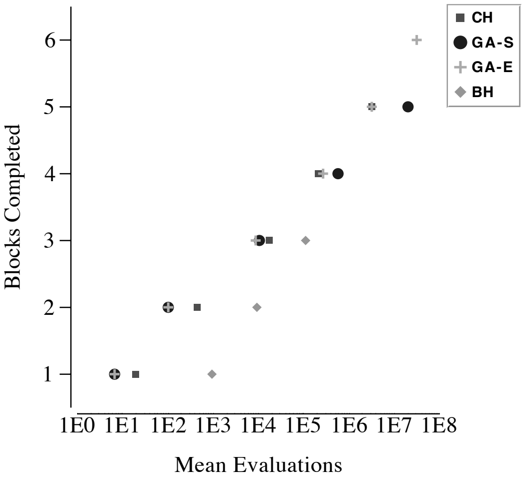


Figure 19. Mean evaluations to solve 10 distributed fully deceptive 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

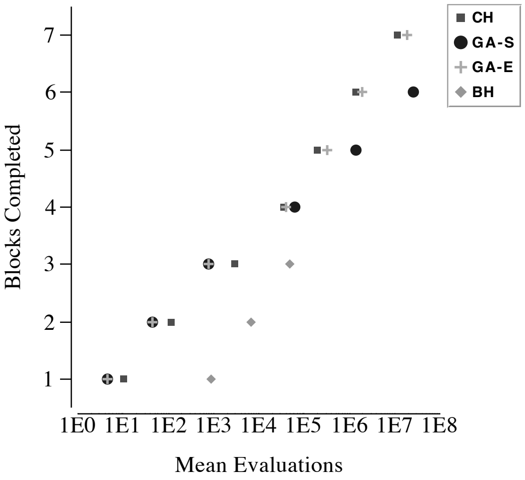


Figure 20. Mean evaluations to solve 15 distributed fully deceptive 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

performance of a GA with that of a hillclimber on this problem [110]. The results obtained there were for a slightly different hillclimbing algorithm (steepest ascent), but qualitatively are in agreement with the relationship between the GA and BH found here.

The BH algorithm performs better than the GA or the CH algorithm for the 2- and 3- state problems. The 4-state problem is considerably more difficult than either of these and here the CH found an optimal TM (producing 13 ones) several times whereas none of the other algorithms did. To find a machine producing 12 ones, BH took on average approximately 7 million evaluations while the CH algorithm took approximately 8 million. GA-S took approximately 45 million and GA-E 14 million. In all cases, the GA performed worse than either of the other algorithms, though the addition of elitism reduced the gap considerably. The performance of BH and CH was very close on the two larger experiments.

3.9.7. Holland's Royal Road

Figures 23 and 24 show the results of the four algorithms on Holland's royal road function that was described in §3.8.6(62). The CH algorithm is a clear winner for the $k = 2$ and $k = 4$ experiments (with 60 bits and 240 bits respectively). On the final experiment, the performance of the two GAs and the CH algorithm are

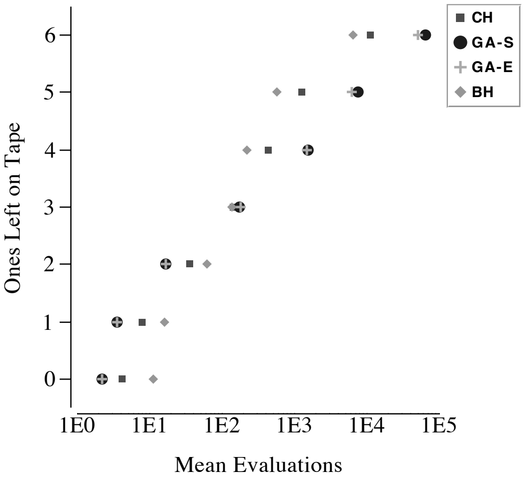


Figure 21. Mean evaluations to find 3-state Turing machines for the busy beaver problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

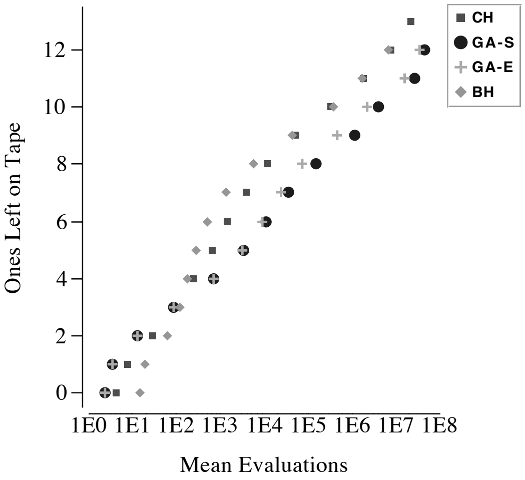


Figure 22. Mean evaluations to find 4-state Turing machines for the busy beaver problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

virtually indistinguishable, with the GAs coming out slightly ahead. In all cases, the BH algorithm does very poorly, which is hardly surprising as one of Holland's design goals was to create a function that would be very difficult to optimize via mutation-based hillclimbing.

The closeness in the results for the GAs and the CH algorithm may be due to the comparatively large size of the problem (960 bits). It is worth noting that the algorithms actually climb *fewer* levels (only 3) in this problem than in the simpler $k = 4$ problem (where they climb 4). The harder problem has 7 levels that could have been climbed whereas the easier problem only has 5. Increasing the parameters on the GAs and CH to allow them to search longer may have revealed a difference.

3.10. Why Does CH Perform so Well?

It is clear that the CH algorithm performs very well on the problems considered. Rather than examine the relative performance of the four algorithms in detail, we will concentrate on the CH algorithm and attempt to discover why it performs so well.

A first impression of the above results might be that the CH algorithm isolates crossover and that it provides a clear demonstration that crossover is a powerful

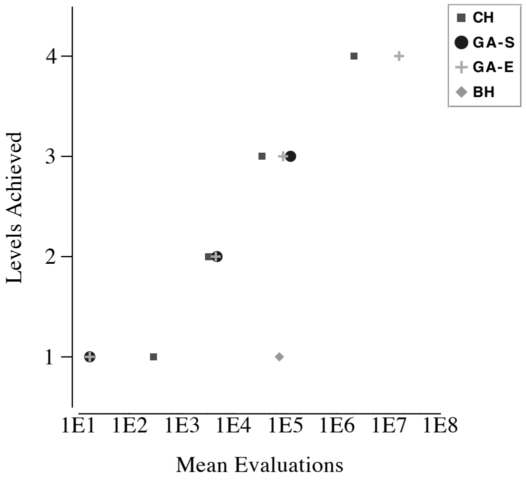


Figure 23. Mean evaluations to climb levels in Holland's royal road with $k = 4$ using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

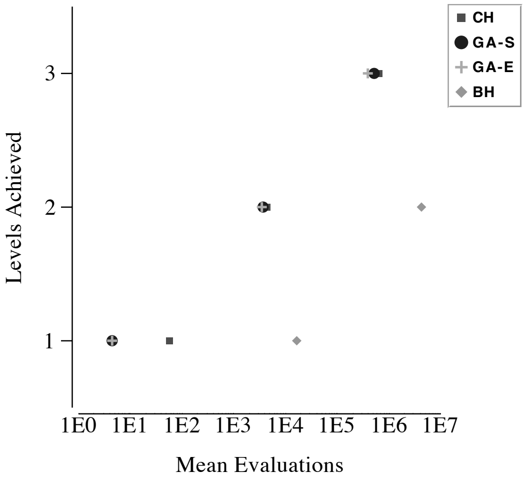


Figure 24. Mean evaluations to climb levels in Holland's royal road with $k = 6$ using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

operator, at times more powerful than mutation. The algorithm was designed to examine crossover in a simple context. To a certain extent this has been achieved, but one loose end remains to be considered. CH is creating new individuals in two ways: through the use of crossover and at random. Initially, two random individuals are created and thereafter a random individual is created each time the algorithm jumps to a new hypercube. How important are these random individuals, and what role do they play in aiding the search? It is simple to examine the number of times a random individual is fitter than any individual seen so far. When this is done, the results are unsurprising. Random creation produces very few highly fit individuals. For example, Table 6 shows the number of times a randomly created individual was responsible for an increase in best fitness over 1000 runs of CH on the 4-state busy beaver problem. In this example, a Turing machine that left seven ones on the tape before halting was discovered on 990 of the 1000 runs, but on only 1 of these was the first such discovery a result of random creation.

To get an idea of the importance of the random individuals as a source of material for crossover to exploit, the CH algorithm used above, CH(10,3,1000), was compared to two variant crossover hillclimbing algorithms, CH(10,3000,0) and CH(10,1,3000). The two new algorithms represent extremes of the spectrum with respect to the random creation of new individuals. CH(10,3000,0) never jumps to a new hypercube

Table 6. The frequency of fitness increases due to random discovery in 1000 runs of the CH algorithm on the 4-state busy beaver problem. The new individual must be the best found so far to gain mention here. The number of times each fitness level was discovered is shown for comparison.

Fitness level	Random discovery	Overall discovery	Fitness level	Random discovery	Overall discovery
0	45	1000	7	1	990
1	133	1000	8	0	786
2	191	1000	9	0	307
3	69	1000	10	0	60
4	24	1000	11	0	13
5	5	1000	12	0	4
6	0	1000	13	0	2

when no acceptable crossover can be found from the current pair. Each search will therefore take place entirely within the hypercube determined by the initial choice of random individuals. When an improving (or equaling) crossover cannot be found from the current pair, the search is concluded. This algorithm will be denoted by CH-NJ (NJ = No Jumps). At the other extreme, CH(10,1,3000) allows at most 1 step to be taken by a pair within a hypercube before a jump to another hypercube is forced. This algorithm will be denoted by CH-1S, (1S = 1 Step). These parameter settings limit all three of these algorithms to 3000 steps via crossover.

Figures 25 to 30 show the results on the largest instance of each of the problems (with the exception of Holland's royal road, which shows the default problem). It is clear that CH-1S is the best of the three algorithms on these instances of these problems. Experiments with other settings of the three parameters of the crossover hillclimbing algorithm (not shown) have provided further evidence that the algorithm performs better the more often it jumps between hypercubes. CH-1S is extreme in this respect, as it jumps after every single crossover. This is remarkable as the first crossover performed after a jump always involves a random individual (the jump is

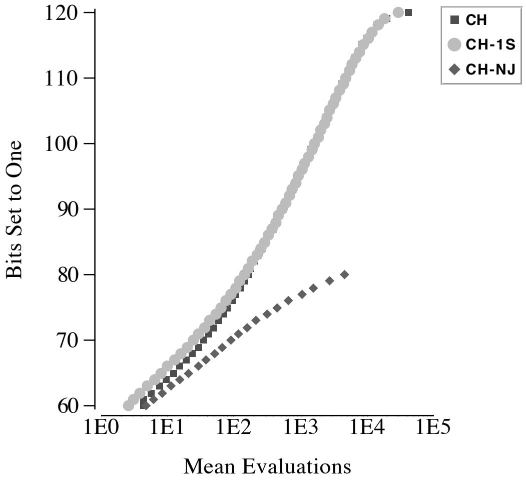


Figure 25. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

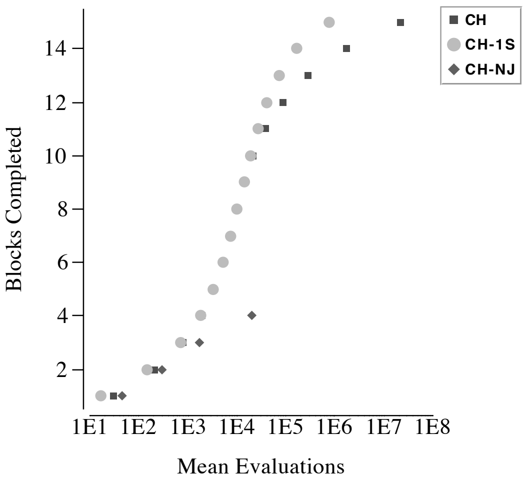


Figure 26. Mean evaluations to complete 15 6-bit fully easy subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

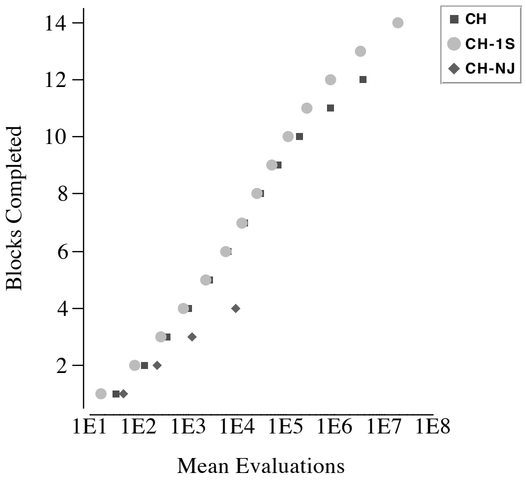


Figure 27. Mean evaluations to complete 15 6-bit fully deceptive subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

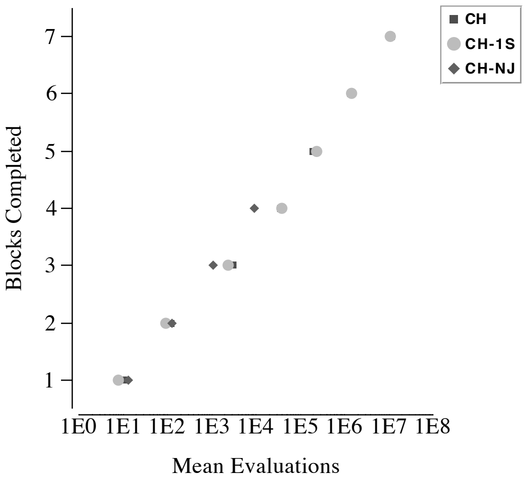


Figure 28. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

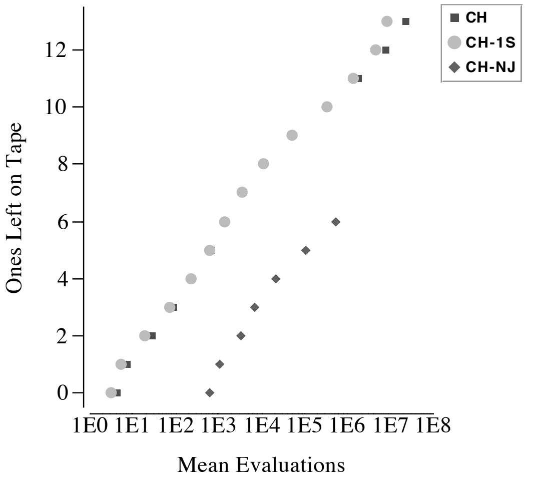


Figure 29. Mean evaluations to find 4-state busy beaver TMs using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

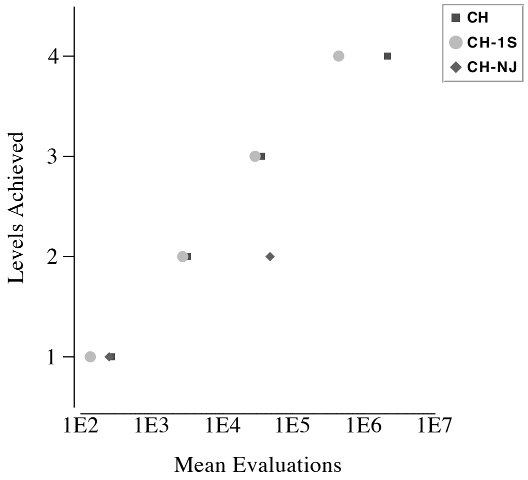


Figure 30. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

performed by replacing one of the current pair by a random individual). As will be seen, crossover involving an individual v and a random individual can be seen as a macromutation of v. For this reason, the CH-1S algorithm is simply performing macromutations that are distributed on the genome in the fashion of two-point crossover.

3.11. Crossover: The Idea and The Mechanics

It is important to consider crossover as playing two roles. These can be called the *idea* and the *mechanics* of crossover. The idea of crossover is that there is some reason to believe that each of the parents has a better than average chance of bestowing better than average genetic material upon its offspring. In a GA, the initial population is simply a collection of random individuals, but soon after this, under the influence of selection, the population is made up of individuals whose performance is better than random. The main point of keeping a population and using crossover is to take better individuals and combine them to potentially form even better individuals. The idea of crossover is clearly present in Holland's early work [102, pages 140, 167 and 180]. The mechanics of crossover is the process by which an attempt is made to implement the idea, for instance via one-point, two-point, uniform or one of the many forms of crossover used for representations other than fixed-length strings. All forms of crossover share a similar idea, but the mechanics vary considerably.

The experiments with various forms of crossover hillclimbing show that crossover may be very useful even in the absence of the idea of crossover. That is, even when there is no reason to believe that *both* parents have an above average chance of contributing above average material to the offspring, crossover may still be useful simply through performing macromutation via its mechanics. When considering whether crossover is useful to a GA, we should attempt to distinguish the gains the algorithm is making using the idea of crossover from those made simply through the mechanics. If there is no additional gain due to the idea of crossover, it may be the case that we would do as well if we discarded the population (and thus the GA) and instead used an algorithm that employed macromutation.

3.12. The Headless Chicken Test

This section presents a simple experiment designed to test whether a GA is making progress via the idea of crossover over that which could be made by simply performing macromutations. To do this, compare the GA with an identical GA that uses *random crossover*. The random crossover operator produces two children, C_1 and C_2, from two parents, P_1 and P_2. To do this it generates two random individuals, R_1 and R_2. The operator then performs a crossover between P_1 and R_1 and keeps one of the children (selected at random), C_1. Then C_2 is produced in an identical fashion, using P_2 and R_2 as parents. The two children are then returned to the GA. This is the only difference between the two algorithms. The random crossover operator is illustrated in Figure 31.

In the algorithm with random crossover, there is no communication between population members since there is no crossover between population members unless, by chance, a randomly generated parent also happens to be present in the population. When one of the individuals involved in a crossover is randomly chosen, the operation is purely mechanical and has nothing of the idea, or spirit, of crossover. In this case, despite the identical mechanical rearrangement, the operation is not crossover. That is, without the idea of crossover, you do not have crossover. To call this operation crossover is similar to calling a headless chicken running around the yard a chicken. Certainly it has many characteristics of a chicken, but an important piece is missing. A further argument that this operation is not really crossover is that two parents are not required. For example, for two-point crossover on strings, simply choose the

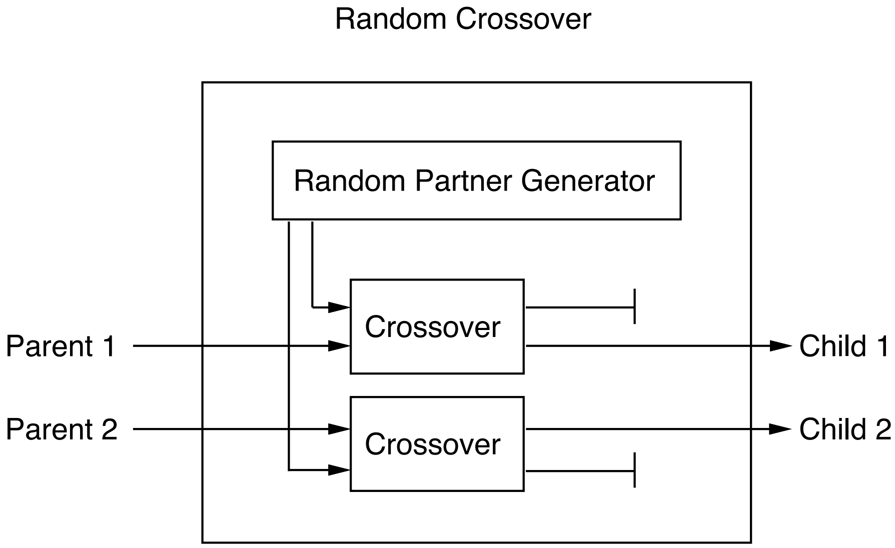


Figure 31. The random crossover operator used in the headless chicken test. When the GA with random crossover passes two parents to the operator, instead of recombining them as a normal crossover would do, two random individuals are created and these are crossed with the parents. One offspring is kept from each of these crosses and these are handed back to the GA. The crossover method used (one-point, two-point, uniform, etc.), is the same as that used in the version of the GA that is being compared to the GA with random crossover.

crossover points and set all the loci between the points to randomly chosen alleles. This is an identical operation and is clearly just a macromutation.

This comparison was performed (1) between GA-S and an identical GA with random crossover (GA-RC) and (2) between GA-E and an elitist GA with random crossover (GA-RCE). Figures 32 to 37 show the result of this experiment on the problem instances used to compare the CH variants. The results of the comparison between GA-E and GA-RCE are not shown (these exhibit identical qualitative results to those of GA-S versus GA-RC). On all instances of all problems, GA-S is a clear winner over GA-RC when the problem contains well-defined building blocks. This is the case with the fully deceptive, fully easy, Holland's royal road and one max problems. On these problems, the idea of crossover is aiding the search, as we would hope.

More interesting is the comparison on the problems where well-defined building blocks do not exist: the busy beaver and distributed fully deceptive problems. On the busy beaver problems, the performance of the GA and GA-RC is hard to distinguish. On the distributed fully deceptive problem, GA-RC actually outperforms the GA.

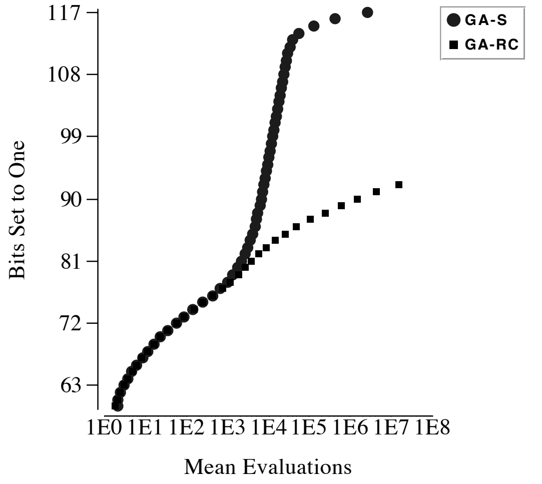


Figure 32. Mean evaluations to solve a 120-bit one max problem using a standard GA (GA-S) and a GA with random crossover (GA-RC).

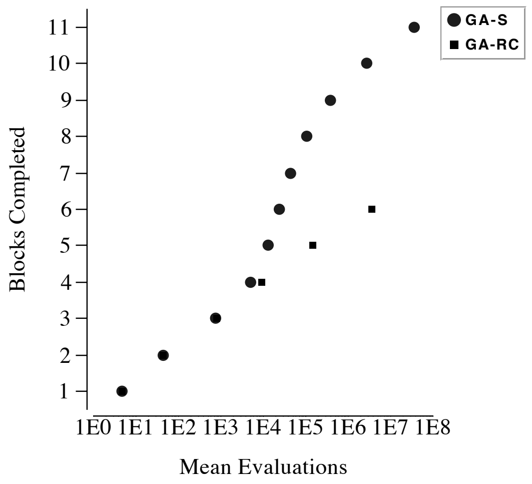


Figure 33. Mean evaluations to complete 15 6-bit fully easy subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

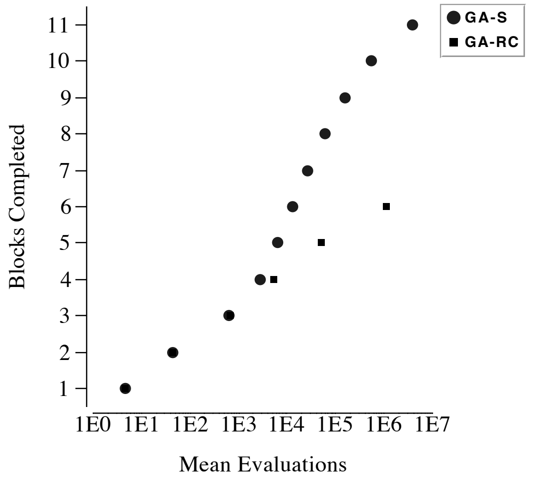


Figure 34. Mean evaluations to complete 15 6-bit fully deceptive subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

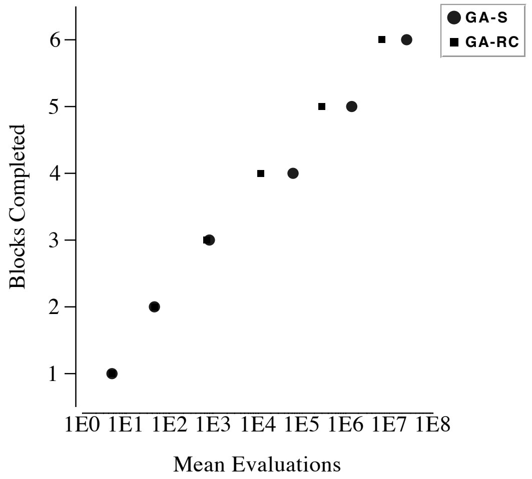


Figure 35. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

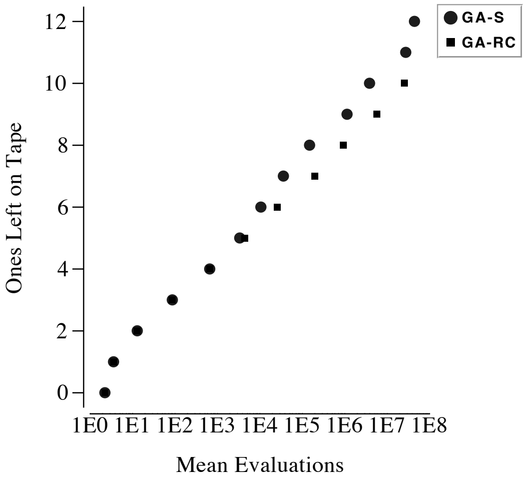


Figure 36. Mean evaluations to find 4-state busy beaver TMs using a standard GA (GA-S) and a GA with random crossover (GA-RC).

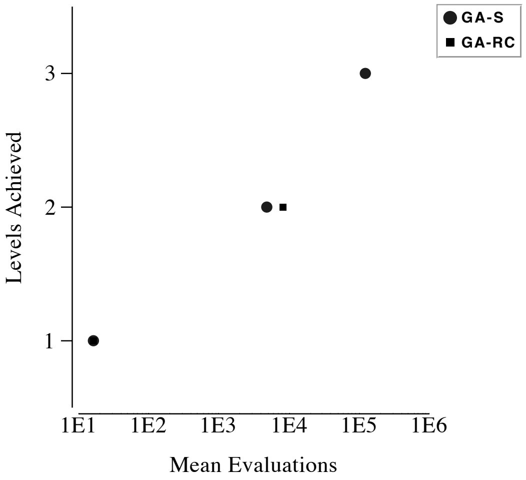


Figure 37. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using a standard GA (GA-S) and a GA with random crossover (GA-RC).

In this case, the GA is not only using an operator (two-point crossover) that, in an informal sense, has a bias that is orthogonal to the structure of the encoding, but it is also drawing its samples from a population. GA-RC is using the same operator but has the advantage of being able to explore more widely. When GA-RC is as good or better than the normal GA on a problem, it suggests that the idea of crossover is doing nothing significant for the normal GA. In such cases it seems safe to conclude that the chosen combination of representation, fitness function and crossover operator does not result in a problem that contains building blocks that the GA is managing to exploit using the idea of crossover. De Jong, Spears and Gordon have also identified problems on which the use of crossover results in worse performance [115].

This does not imply that there is no other representation that would contain exploitable building blocks, or that another crossover operator might not do better at exploiting the building blocks that do exist (if any), or that it is not possible to modify the GA in some way so that it does exploit the building blocks (e.g., by using a larger population, a different fitness function, or adopting measures to maintain diversity). This is a conclusion about a single crossover operator, a single representation and a single algorithm, and may well not apply if any of these are altered. The test examines a particular set of choices and nothing more. In particular, the conclusion that a GA is not suitable for a problem, based on a failed headless chicken test, is in no way justified. If a GA fails the headless chicken test (i.e., it does not outperform a

GA with random crossover), the GA is not making gains from crossover above those that could be made via explicit macromutation. If this is the case, it is not clear why one would bother to maintain a population or, consequently, use a GA on *this combination* of problem, representation and fitness function.

The experiment suggests an explanation for GAs that appear to make good use of crossover early in a run, but not thereafter. It may be the case that the combination of the representation and fitness function does not have building blocks that can be exploited by the crossover operator in question. In this case, crossover may make improvements simply through macromutations at the start of the run and then, as is widely known, become increasingly less useful as the population converges. Crossover *appears* to be useful (since disabling it results in worse performance), but in fact, its usefulness is merely a consequence of the macromutations it is performing.

3.13. Macromutational Hillclimbing

The success of CH-1S and the observation that it is simply performing macromutation prompted the implementation of two hillclimbers that use a macromutational operator and the navigation strategy of BH. In the mutation operator of the first, (BH-MM), the genome is viewed as a ring. Two distinct points on the ring are selected uniformly at random and the loci in the smaller section are set to random alleles (including those that are already present).

The smaller section is chosen for mutation as observation revealed that CH was making the great majority of its improving steps when it modified less than half the genome. The result of this mutation operator is exactly what a crossover with a random individual would produce if the child that inherited most from the non-random parent was retained. The operator distributes the mutations in the same way that two-point crossover does. This hillclimber is very similar to CH(1,1,∞). In the second hillclimber, the macromutation operator decides on a number of loci to set randomly (as in BH-MM) but then distributes these mutations at random through the string. The behavior of BH-DMM is similar to that which would be obtained via crossover with a random string using uniform crossover.

This operator was described in terms of strings for simplicity, but it is more general. Any representation that can be viewed as containing a number of independently adjustable components could have a similar operator designed for it. Such representations are very common, and were described briefly in §2.6(30).

These two hillclimbers were compared to CH-1S and GA-E. In every case but the busy beaver problems, one of the macromutational hillclimbers always proved the best algorithm. Once again, the results depended on whether well-defined building blocks existed. When they did, BH-MM outperformed BH-DMM by orders of magnitude. When they did not (on the distributed fully deceptive problem), BH-DMM performed an order of magnitude better than BH-MM. On every problem, BH-MM achieved the highest levels of performance at least one order of magnitude faster than GA-E, and in one case (15 fully easy subproblems) it required over 3 orders of magnitude fewer evaluations to achieve the highest level reached by GA-E. Typically, the hillclimbers would also reach levels of performance that were never achieved by the GA. The results for the six problems instances are shown in Figures 38 to 43.

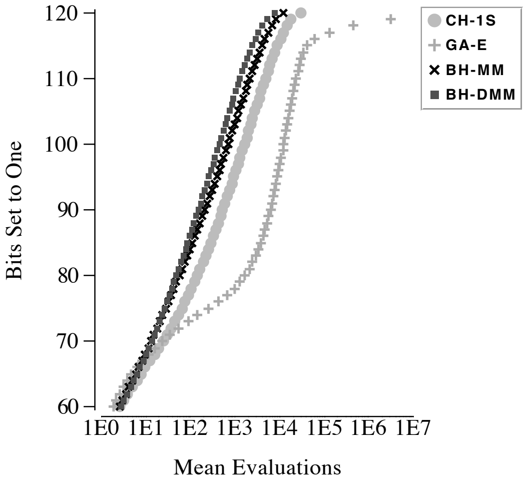


Figure 38. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

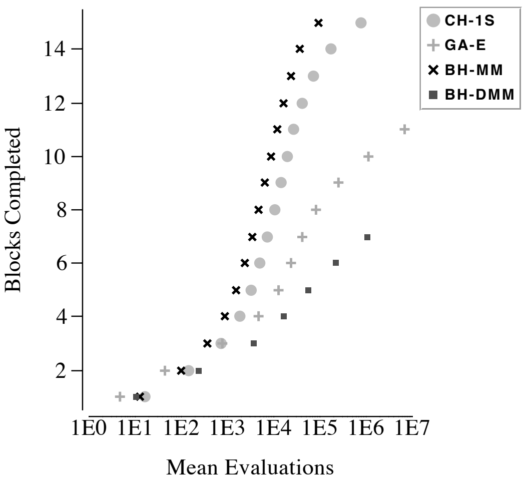


Figure 39. Mean evaluations to complete 15 6-bit fully easy subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

3.14. Summary

The traditional method used to assess whether crossover is useful to a GA is to compare the GA with crossover to a GA without crossover. This chapter argues that if the GA with crossover performs better, the conclusion that crossover is therefore useful (i.e., that it is facilitating the exchange of building blocks between individuals) is not justified. A more informative method of assessing the worth of crossover to

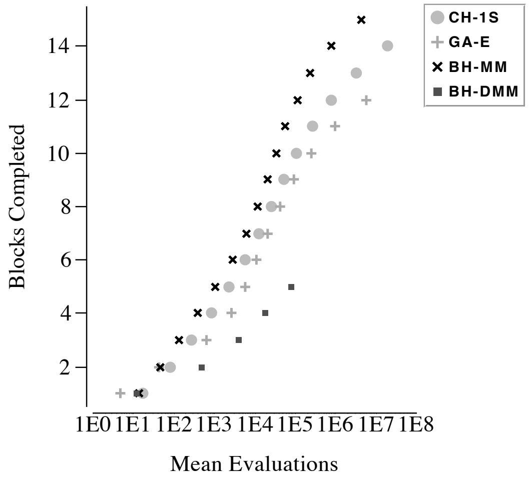


Figure 40. Mean evaluations to complete 15 6-bit fully deceptive subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

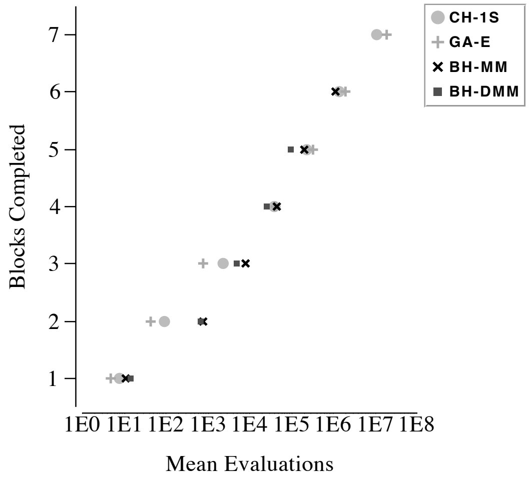


Figure 41. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

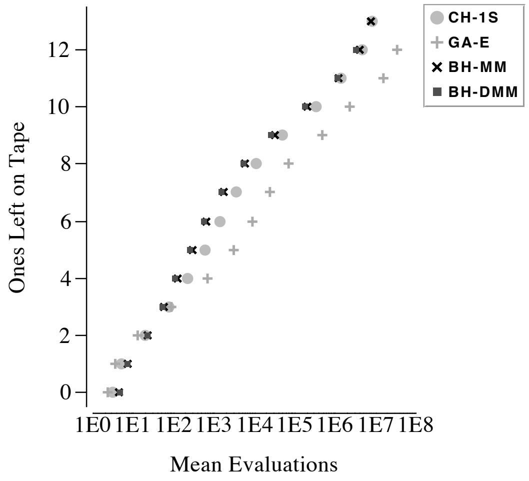


Figure 42. Mean evaluations to find 4-state busy beaver TMs using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

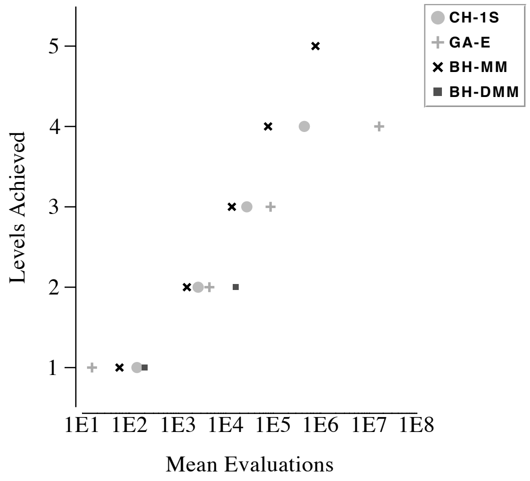


Figure 43. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

a GA was presented. This compares the normal GA with a GA that uses random crossover. Such a GA dispenses with the idea of crossover while maintaining the mechanics. This makes it possible to see what advantage the standard GA is gaining from the idea of crossover over and above what it could be gaining from the mechanics alone. In some cases, when well-defined building blocks are not present, the GA may actually perform *worse* with normal crossover than a GA with random crossover as a result of its use of a population. This test gives an indication of when building blocks (that are exploitable by a GA with a given crossover operator) exist in a representation.

The major reason for the maintenance of a population in a GA is to allow the communication of information between individuals via crossover. When testing whether it is worth bringing a population and crossover to bear on a problem, the null hypothesis should be that the GA with crossover does not outperform the GA with random crossover, *not* that the GA with crossover outperforms the GA without crossover. If this comparison indicates that the idea of crossover is not producing significant gains, the use of a GA is probably not warranted. In these cases, there are simpler algorithms that use the macromutational mechanics of crossover and do not maintain a population, and these can easily outperform the GA. Several such algorithms, based on crossover hillclimbing or macromutational hillclimbing were presented. This point was also made by Fogel and Atmar [88]. On the other hand, when building blocks do exist, these algorithms (especially those that use macromutation exclusively) still outperformed the GA on virtually every instance of every problem addressed. These results support Eshelman and Schaffer's belief that the "niche" in which crossover gives a GA a competitive advantage may be quite small [89]. Actually, the situation may be worse than they feared, as a macromutational hillclimber easily outperforms a GA on Holland's royal road, which has the properties that Eshelman and Schaffer ascribe to problems residing in crossover's niche. The niche becomes smaller if we insist that the idea of crossover, in addition to the mechanics, be responsible for good performance. In the context of function optimization, it is clear that care must be taken if one intends to make effective use of a population, crossover, and a GA.

CHAPTER 4

Reverse Hillclimbing

4.1. Introduction

There are many algorithms that can be informally described as hillclimbers. Though simplistic, many hillclimbers prove surprisingly efficient in some settings. Slight enhancements can result in algorithms which are amongst the most useful (e.g., simulated annealing). Despite the simplicity of hillclimbing algorithms, it is difficult to accurately compare these algorithms to each other. To compare the performance of two hillclimbers, the simplest method is to run them both a large number of times. The results give a statistical indication of which algorithm is superior. This chapter introduces a new technique, *reverse hillclimbing*, which, in some cases, allows a much more direct comparison. This technique is used to compare four hillclimbing algorithms on three instances of two problems.

The method also makes it possible to compute statistics regarding basins of attraction in a landscape graph. Given a peak on a landscape, reverse hillclimbing computes the exact size of the basin of the point under a hillclimbing algorithm and the exact probability that the peak will be found via a single hillclimb. An attempt to obtain similar statistics using hillclimbing to form estimates would probably consume inordinate amounts of time. For example, a typical global optimum in the 4-state busy beaver problem has a basin of attraction under steepest ascent hillclimbing of size approximately 5000. The probability of locating such a peak from a randomly started hillclimb is usually about 1 in 15 million. Thus one would expect to perform 15 million hillclimbs before stumbling on the peak even once. This would net a few of the 5000 vertices in the basin. Obtaining a suitably reliable estimate (let alone the exact number) of the basin's size would require a far greater number of hillclimbs. Reverse hillclimbing can provide exact answers to these questions, and others, in under a minute with very limited computational requirements. To my knowledge, none of the information provided by reverse hillclimbing could be obtained exactly in the past without an exhaustive enumeration of the vertices in the landscape. Perhaps as a result, there are few results that deal with basin sizes.

Given a vertex v in the graph of a landscape $\mathcal{L} = (\mathcal{R}, \phi, f, \mathcal{F}, >_{\mathcal{F}})$, reverse hillclimbing provides exact answers to the following questions about the basin of attraction of v, $B_\phi(v)$:

- What are the elements of $B_\phi(v)$?
- By how many paths can each element of $B_\phi(v)$ ascend to v? What are the lengths of these paths and what is the expected number of evaluations along each path?
- What is the probability that a single hillclimb, started from a randomly chosen vertex, will ascend to v?
- How many elements of $B_\phi(v)$ can ascend only to v?

When reverse hillclimbing is applied to a large number of peaks on a landscape, the collection of peak fitnesses, basin sizes and ascent probabilities, allows the examination of the relationships between

- peak height and basin size,
- peak height and the probability of peak discovery, and
- basin size and probability of peak discovery.

4.2. Local Search Algorithms

Given the graph $G_{\mathcal{L}} = (V, E)$ of a landscape $\mathcal{L} = (\mathcal{R}, \phi, f, \mathcal{F}, >_{\mathcal{F}})$, *Local search algorithms* are those algorithms whose behavior is shown in Algorithm 4.1. This defines a broad class of algorithms that can be thought of as taking a random walk using their operator. The fitnesses encountered in such a walk will depend entirely on the operator used. If $\mathcal{R} = \{0,1\}^n$ and β, the bit-flipping operator described in §2.6.1(31), is used, then any point can be reached from any other. Local search algorithms that are more useful than random search on a given class of problems will use operators that tend to produce higher fitness points from lower.

```
choose a vertex $v \in V$;
while (not done) {
    Use $\phi$ to generate $w \in N_\phi(v)$;
    $v \leftarrow w$;
}
```

Algorithm 4.1 Local search algorithm.

The initial vertex is commonly chosen by uniform random selection from the representation space. There are many other possibilities. For example, the search may commence from a vertex generated by another search method, or the initial vertex may be chosen according to some belief about the nature of the landscape, or may be a starting configuration that is given by the statement of the problem. The second step, generating $w \in N_\phi(v)$ also has many variants. In the algorithms which we will consider, ϕ will employ β to generate $N \subseteq N_\beta(v)$, one element of which, w, will be selected. As examples, some operators will use β to generate $N = N_\phi(v)$, the entire β-neighborhood of v and then select the fittest. Others will use β to generate a fixed number of β-neighbors of v, or generate β-neighbors until a vertex fitter than v is located or some limit is reached. In the final step, the algorithm either terminates or loops for another iteration. The decision of when to terminate a search may be according to some maximum number of evaluations, the achievement of a desired fitness level, the detection of a local maximum, the result of running out of time for the search, or some other method. The above description also deliberately makes no mention of the return value of the algorithm. Typically these algorithms keep track of the best element of $\mathcal{R}$ encountered during the search and return that. Algorithms that never select a w such that $f(w) <_\mathcal{F} f(v)$ can simply return v when they are done.

4.2.1. Iterated Local Search Algorithms

A simple enhancement to a local search algorithm is an iterated version which repeats the original local search algorithm some number of times. These algorithms are often employed on problems in which the initial vertex is chosen uniformly at random, though there are important exceptions [116].

4.3. Hillclimbing Algorithms

The definition of local search algorithms provides a framework for the definition of hillclimbing algorithms. A hillclimbing algorithm is a local search algorithm that always selects a w in step 2 above such that $f(w) \geq_{\mathcal{F}} f(v)$. Under this definition, "hillclimbing" means that the algorithm never goes downhill. One argument for including algorithms that sometimes select a w such that $f(w) = f(v)$ is that every algorithm which makes strictly increasing moves can be simply modified to also allow moves to vertices of equal fitness. Also, this definition of a hillclimber is in accordance with Minsky's early description of hillclimbers as climbing hills (obviously), but also as getting trapped at "false peaks" and as potentially wandering on "mesas" of equal fitness [25].

Dividing local search algorithms on the basis of the relative fitnesses of v and w leads to three classes, $LSA_{<=>}$, $LSA_{>=}$ and $LSA_{>}$, with $LSA_{>} \subset LSA_{>=} \subset LSA_{<=>}$. These are depicted in Figure 44. Hillclimbers are those algorithms found in the two inner circles. This taxonomy will allow easy discussion of a number of algorithms and delineation of the class algorithms upon which reverse hillclimbing can be applied. This chapter explores the characteristics of four hillclimbing algorithms from $LSA_{>}$ on a number of problems. These are, Any Ascent, Least Ascent, Median Ascent and Steepest Ascent. Figures 45 to 48 shows the landscape corresponding to each of the operators of these algorithms. In these landscape graphs, $\mathcal{R} = \{0, 1\}^n$, f, $\mathcal{F}$, and $>_{\mathcal{F}}$ are held constant.

4.3.1. Any Ascent Hillclimbing

The *any ascent* hillclimbing algorithm (AA), employs an operator that uses β to generate β-neighbors of v until it finds a w such that $f(w) >_{\mathcal{F}} f(v)$. Of course if such a β-neighbor does not exist, the algorithm should not loop indefinitely. This can be avoided by keeping track of the neighbors generated by β, or by using some limit on the number of β-neighbors that can be tested. The first solution, the one used in this chapter, also makes it possible to avoid evaluating β-neighbors multiple times. The second solution results in an algorithm whose sibling in $LSA_{>=}$ is the BH algorithm of Chapter 3.

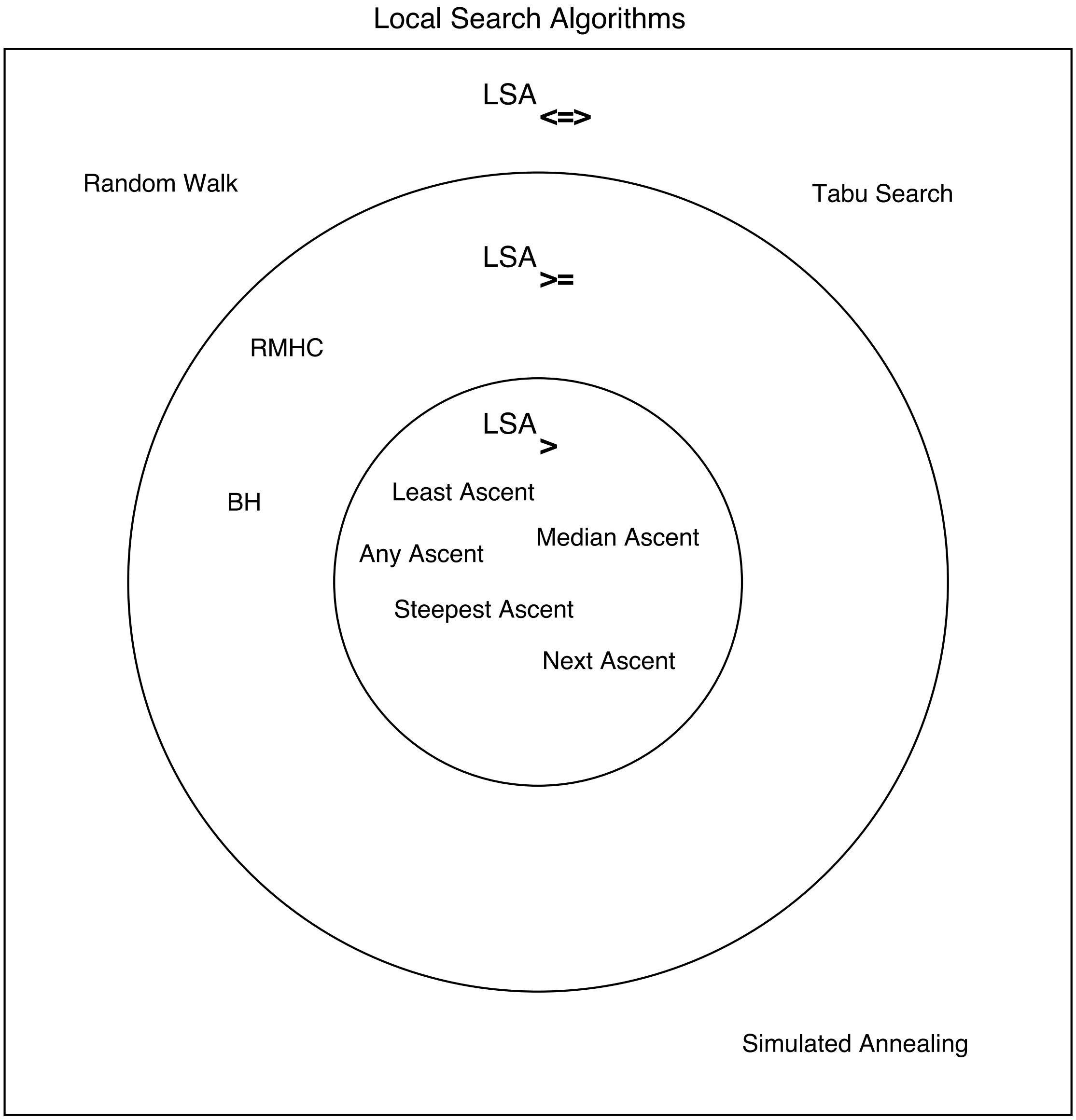


Figure 44. A division of local search algorithms according to the types of fitness moves they make. Hillclimbing algorithms comprise the inner two circles. The RMHC and BH algorithms are described in Chapter 3.

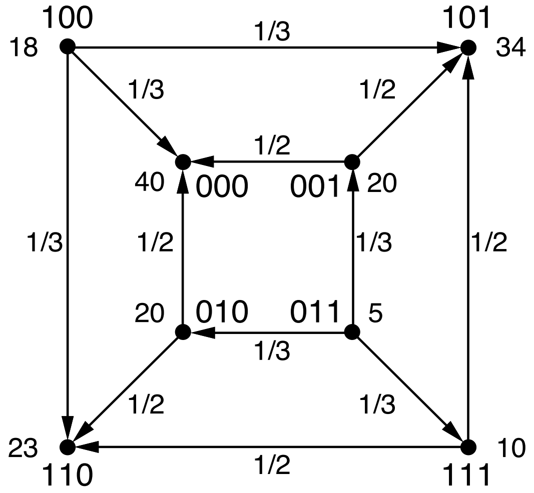


Figure 45. The landscape produced by the *any ascent* operator given a representation space $\{0,1\}^3$ and a hypothetical fitness function. Vertices are labeled with fitnesses and edges with transition probabilities.

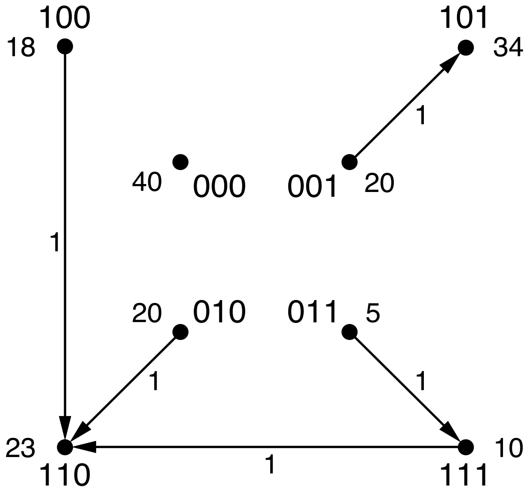


Figure 46. The landscape produced by the *least ascent* operator given a representation space $\{0,1\}^3$ and a hypothetical fitness function. Vertices are labeled with fitnesses and edges with transition probabilities.

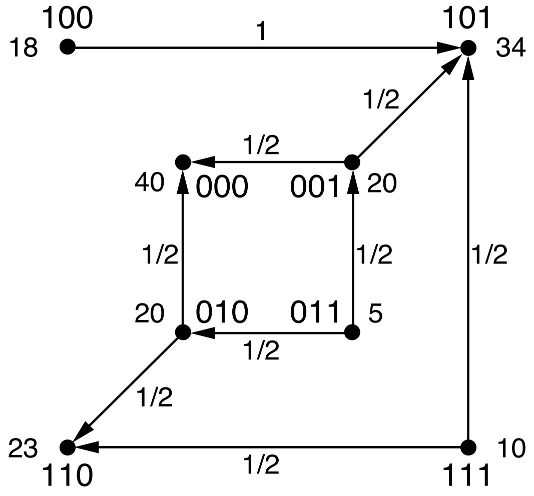


Figure 47. The landscape produced by the *median ascent* operator given a representation space $\{0,1\}^3$ and a hypothetical fitness function. Vertices are labeled with fitnesses and edges with transition probabilities.

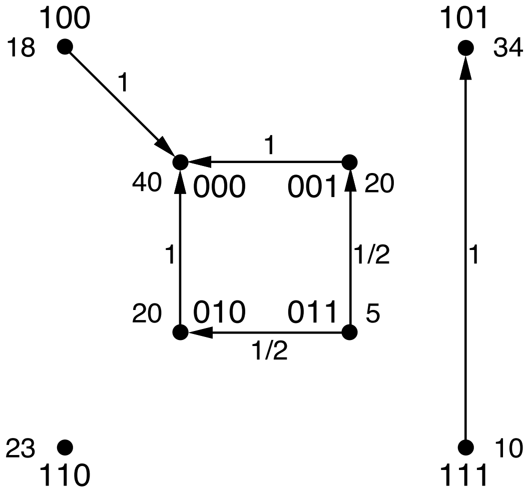


Figure 48. The landscape produced by the *steepest ascent* operator given a representation space $\{0,1\}^3$ and a hypothetical fitness function. Vertices are labeled with fitnesses and edges with transition probabilities.

4.3.2. Least Ascent Hillclimbing

The *least ascent* hillclimbing algorithm (LA), employs an operator which uses β to generate all of $N_\beta(v)$ and sets w to an element of $N_\beta(v)$ with least fitness subject to the condition $f(w) \geq_{\mathcal{F}} f(v)$. In other words, from the subset of $N_\beta(v)$ with fitness greater than v the least fit element is selected. This algorithm therefore takes a conservative approach to hillclimbing, never moving uphill more than necessary. There may be several equally fit elements of $N_\beta(v)$ that could be selected. In this case, one is chosen uniformly at random.

4.3.3. Median Ascent Hillclimbing

The *median ascent* hillclimbing algorithm (MA), employs an operator which uses β to generate all of $N_\beta(v)$ and sets w to an element of $N_\beta(v)$ with median fitness amongst those with fitness greater than $f(v)$. This algorithm therefore takes a middle-of-the-road approach to hillclimbing, moving to the middle of all the uphill neighbors. There may be an even number of fitter β-neighbors. In this case, one of the two medians is chosen uniformly at random. Vertex 011 in Figure 47 shows an example of this.

4.3.4. Steepest Ascent Hillclimbing

The *steepest ascent* hillclimbing algorithm (SA), employs an operator which uses β to generate all of $N_\beta(v)$ and sets w to an element of $N_\beta(v)$ with greatest fitness. There may be several maximally fit elements of $N_\beta(v)$ that could be selected. In this case, one is chosen uniformly at random. Vertex 011 in Figure 48 shows an example of this.

4.4. The Reverse Hillclimbing Algorithm

This section describes the basic reverse hillclimbing algorithm, shows how it may be enhanced to obtain more information, and discusses some implementation issues.

4.4.1. The Basic Algorithm

The reverse hillclimbing algorithm is conceptually quite simple. Algorithm 4.2 shows a recursive function that collects the vertices in the basin of attraction of a vertex v into a set S (initialized to the empty set) given a hillclimbing algorithm, $H \in LSA_>$, that uses an operator ϕ. When the recursion terminates, the set S will contain the basin of

```
basin_size(v)
VERTEX v;
{
    $S = S \cup \{v\}$;

    for each $(w | v \in N_\phi(w)$ and $f(w) <_\mathcal{F} f(v))$
        basin_size(w);
}
```

Algorithm 4.2 The basic reverse hillclimbing algorithm. S, an external variable, is initialized to the empty set before the initial invocation of the function. S will contain the vertices in the basin of attraction when the recursion terminates.

attraction of v for operator ϕ. The above function is not an efficient implementation for many reasons Some of these are discussed in §4.4.2 and Appendix A. There are several points that should be noted about this algorithm:

- Given a representation, $\mathcal{R}$, and fitness function, the basin of attraction of an element of $\mathcal{R}$ depends on the operator in use. This can be seen by considering the basins of attraction for the vertex 110 in Figures 45 to 48.

- Assuming the set of vertices that may have v as a neighbor can be identified without search, reverse hillclimbing can be implemented so that it never examines a vertex in the space that is more than a single step from a vertex in the basin of attraction. When vertices outside the basin of attraction are examined, they do not become the subject of a subsequent recursive call. This leads immediately to an upper bound on the number of vertices examined by the algorithm of $n|B_\phi(v)|$ where n is the maximum in-degree in the landscape and $|B_\phi(v)|$ is the size of the basin of attraction. In practice, the number of elements of $\mathcal{R}$ evaluated will be far smaller than this for two reasons: first, it is common for several less fit neighbors of a vertex to be able to re-ascend to that vertex (for instance in AA they all can) and second, the sets of downhill neighbors of two vertices will often not be disjoint.

- The function *basin_size* may be called more than once on the same vertex. To see why this is true, consider Figure 49 which shows two paths to the same vertex in a one max problem.

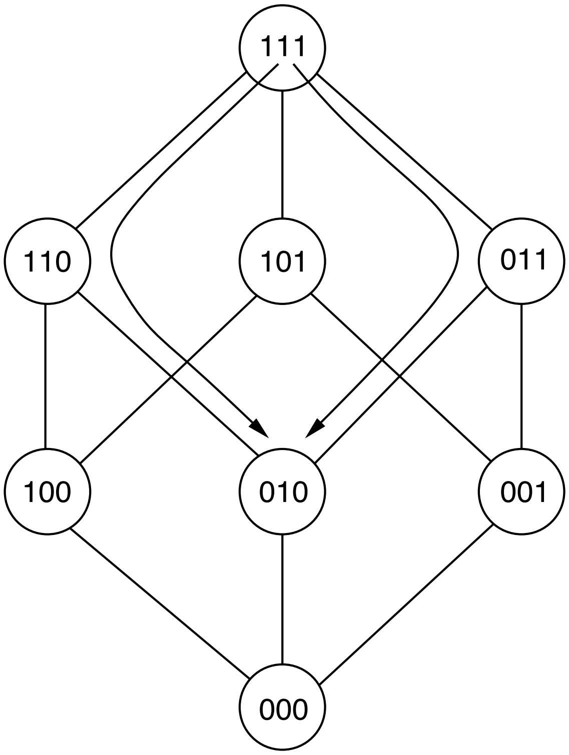


Figure 49. Two downhill trajectories when performing reverse hillclimbing on a one max problem with strings of length three. The arrows illustrate how a vertex (in this case 010) can be encountered more than once. In the basic algorithm, this vertex would be the subject of two recursive calls to the *basin_size* function.

4.4.2. Augmenting the Basic Algorithm

The basic reverse hillclimbing algorithm given above only produces the vertices in the basin of attraction of a vertex. This algorithm may be augmented to provide the additional details of the basin and the vertices in it that were mentioned in the introduction: ascent probabilities, number and lengths of ascending paths, and expected evaluations to peak discovery.

Because the algorithm will typically encounter vertices in the basin on more than one occasion, it is necessary to store partial information about vertices until they have been encountered for the final time, at which point they can be output or their properties can be added to accumulating statistics about the basin. A straightforward solution to this problem is to use a hash table to implement the set S above to keep track of the details of vertices that have been encountered. This allows the maintenance of details of probability of ascent, number of ways to ascend and expected comparisons to ascend for each vertex in the basin. When the function *basin_size* discovers a vertex that could ascend to the current argument, it checks the hash table to see if this vertex has already been encountered. If so, it updates the data structure containing the information for the vertex. If not, it creates a new data structure and inserts it into the hash table. A rough outline of how each statistic is kept follows:

- If a vertex v_1 can ascend along some path to the original vertex v with probability p, and it is discovered that v_2, a less fit neighbor of v_1, can ascend to v_1 with probability q, then pq is added to the overall probability that v_2 can ascend to v. Note that the algorithm may encounter v_1 again (by finding another downhill re-ascending path to it), and this will cause the probabilities associated with both v_1 and v_2 to be incremented again. When the recursion is finished, every downhill path will have been taken, and the probabilities associated with the vertices in the basins will be correct.

- It is easy to keep track of the number of paths by which a vertex may ascend to the original vertex. Simply keep a count of the number of times each vertex in the basin is encountered during the recursion (including the first, in which the vertex is not found in the hash table).

- If a vertex can only ascend to a single peak under a given hillclimbing algorithm, the vertex will be said to be *owned* by the peak that is reached. To calculate how many of the vertices in the basin of attraction are owned by the peak, look at the final ascent probabilities. Those vertices with probability one are owned.

- Calculating the expected number of vertex fitness evaluations it will take a vertex to reach the peak is similar to the calculation of ascent probability. Information is passed down through the basin as paths are traversed. Three of the four hillclimbers, LA, MA and SA, use β to produce all the β-neighbors, so their expected evaluations per uphill step is simply the number of β-neighbors at each step. To calculate the expected number of evaluations for an uphill step of AA, it is necessary to determine the number of uphill β-neighbors that exist at that vertex. If we let n be the number of β-neighbors and u be the number of uphill β-neighbors, then the expected number of evaluations before the vertex is left is given by

$$\begin{aligned} E(evaluations) &= \sum_{i=1}^{\infty} i \left(\frac{n-u}{n} \right)^{i-1} \frac{u}{n} \\ &= \frac{u}{n} \frac{1}{\left(1 - \frac{n-u}{n}\right)^2} \\ &= \frac{u}{n} \frac{1}{\left(\frac{u}{n}\right)^2} \\ &= \frac{n}{u} \end{aligned}$$

The second line of this derivation follows from the equation obtained by differentiating the identity $\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots$.

This list is merely a brief overview of how further details of the basin of attraction and its vertices may be obtained. It should provide enough information for the algorithm to be implemented. Further discussion of implementation is provided in Appendix A.

4.5. An Important Tradeoff

There is an important tradeoff between hillclimbing and reverse hillclimbing. These algorithms can both be used to provide useful information about a landscape. Typically what one algorithm can do well, the other cannot. For example, on a large landscape with a single peak whose basin of attraction is the entire landscape, it is impractical to calculate the size of the basin using reverse hillclimbing. The basin may have more vertices than there are atoms in the universe, in which case it would would be far more efficient to arrive at this conclusion, with some (estimable) probability of being wrong, by undertaking a million hillclimbs from random vertices and observing that they all arrived at the same place. If, however, a peak has been found through some means and one wishes to calculate the probability of this occurring (for instance to test an hypothesis that the algorithm that found the peak was merely lucky), using hillclimbing to form an estimate of the peak's basin size and probability of attraction can be equally time-consuming. Reverse hillclimbing may be able to provide the answer rapidly.

The reverse hillclimbing algorithm has the pleasing property of becoming more effective as the landscapes on which it is set to work become more difficult to search. As problems get harder and bigger, it may be that reverse hillclimbing can *only* practically be employed to find details of the hardest to locate areas of these hardest problems. Naturally, the algorithm does not help to identify these areas before setting to work on exploring them.

4.6. When can a Hillclimber be Reversed?

I have defined hillclimbers to be the algorithms in $LSA_{>=}$, but this chapter deals with four hillclimbers that are members of $LSA_{>}$, a proper subset of $LSA_{>=}$. This raises some questions:

- **Can reverse hillclimbing be applied to all algorithms in $LSA_{>}$?** That is, can it be applied to all hillclimbing algorithms that only take uphill steps? I believe the answer to this question is that it can in theory. In practice, there are three reasons why this may be difficult or impractical:

    1. The neighborhood of vertices in the landscape may be too large to completely examine. In the extreme, the neighborhood of each vertex includes every other vertex. This is the case with the mutation operators of GAs. It is not practical to perform reverse hillclimbing on this landscape for exactly the same reason that it is not practical to perform SA hillclimbing on it.
    2. The calculation of ascent probabilities might be difficult for some algorithms.
    3. The hillclimbing algorithm may be relatively difficult to reverse, as in the case of *next ascent*. This does not prevent reverse hillclimbing from being done, but will make its implementation more challenging.

- **Can reverse hillclimbing be applied to algorithms in $LSA_{>=} - LSA_{>}$?** That is, can one perform reverse hillclimbing on a hillclimbing algorithm that potentially makes moves to vertices of equal fitness? The simplest answer is that this appears difficult, since these algorithms may spend an arbitrary amount of time exploring a plateau. If the reverse hillclimbing algorithm was provided with analytically derived results to deal with the expected results of this behavior, the situation might be improved, but even that seems doubtful. The amount of information that would need to be provided to the reverse hillclimbing algorithm so it could avoid infinite loops would likely be enough to solve the original problem. These algorithms might be analyzed completely using techniques such as Markov chains, though in practice there will be far too many states in such a Markov process to make study of even moderate landscapes feasible.

All is not lost however, as it is still possible to determine the basin of attraction of a peak. The hash table-based reverse hillclimbing algorithm can be modified to include vertices of equal fitness in the basin, but not attempt to calculate the number of ascent paths or the probabilities of these paths.

- **Can reverse hillclimbing be applied to algorithms in $LSA_{<=>} - LSA_{>=}$?** That is, can the method be applied to local search algorithms that potentially take downhill steps? The answer appears to be no. As above, it may still be possible to determine the basin of attraction (without knowing the number of paths, their probabilities and their expected evaluations etc.), but to get the full complement of information, it appears that other methods would need to be used, and it is not clear what these might be.

4.7. Test Problems

This chapter demonstrates how reverse hillclimbing can be used to compare hillclimbing algorithms via examination of the basins of attraction that arise from instances of two search problems. The first is the busy beaver problem, and the second is the problem of locating high fitness vertices on NK landscapes.

4.7.1. NK Landscapes

The NK landscapes were introduced by Kauffman [117, 118] as a simplistic model of systems with many interacting parts, particularly the complex epistatic interactions found in genomic systems. An NK landscape has two important parameters: N and K. N is the number of dimensions in the landscape, each dimension having some fixed number of possible values. Attention has been focused almost exclusively on binary dimensions, and this is also the approach of this chapter. A vertex in an NK landscape can therefore be represented by a binary string of length N. The N positions are often referred to as loci.

The fitness of a vertex in an NK landscape is computed in an additive fashion. Each of the N loci makes a contribution to the overall fitness. These are then summed and divided by N to obtain the fitness of the vertex. The variable K controls the degree to which the fitness contribution of each locus is affected by the binary values found at other loci. K is the average number of other loci that affect the fitness contribution of each locus. If $K = 0$, the fitness contribution of each locus is independent of every other locus, and the optimal value for each locus can be determined by simply comparing the overall fitness of two points—one with the locus set to 0 and the other with the locus set to 1. If $K = N - 1$, every locus is affected by every other locus and changing the value assigned to a locus also changes the fitness contribution of every other locus.

A simplistic way to implement an NK landscape requires filling an N by 2^{K+1} array with real numbers chosen uniformly at random from the interval $[0{\cdot}0, 1{\cdot}0]$. A simple method of choosing influencing loci is to consider the N loci to form a ring, and let the $\lfloor K/2 \rfloor$ loci to the left and $\lceil K/2 \rceil$ loci to the right of each locus influence that locus. An NK landscape with $N = 4$ and $K = 2$ and an example fitness calculation is shown in Figure 50. Neighborhood in an NK landscape is defined by the operator β defined in §2.6.1(31). In general, one cannot store $N2^{K+1}$ random numbers and so this simplistic method must be replaced with something more sophisticated, and there are several ways in which this can be done. This presentation of NK landscapes is deliberately very brief, as more detailed presentations are easily available [117, 118].

If N is fixed and the value of K is allowed to vary, the set of landscapes that are produced will range from almost certainly unimodal (containing a single global maximum) for $K = 0$, to completely random (meaning that the fitness of a vertex is completely uncorrelated with that of any of its N one-mutant neighbors) for $K = N - 1$. The NK landscapes are thus said to be *tunably rugged.* Because there are aspects of some NK landscapes that are amenable to analytical investigation and because there are other aspects of these landscapes that have been studied empirically [117, 118], these landscapes provide a good testbed for reverse hillclimbing studies.

4.7.2. Busy Beavers

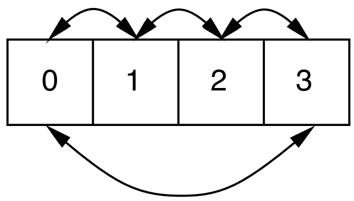
The busy beaver problem is described in §3.8.5(60).

4.8. Experiment Overview

Reverse hillclimbing makes it possible to study many things. These include details about particular landscapes or ensembles of landscapes, properties of hillclimbing algorithms and relationships between hillclimbing algorithms. The experiments of

Neighborhood	Locus 0	Locus 1	Locus 2	Locus 3
000	.18	.04	.73	.91
001	.42	.77	.35	.11
010	.68	.31	.04	.89
011	.91	.17	.25	.70
100	.93	.12	.73	.53
101	.59	.64	.82	.77
110	.19	.94	.38	.21
111	.43	.49	.62	.55

Influencers



Fitness calculation for string 0110

Locus	Neighborhood	Fitness
0	001	.42
1	011	.17
2	110	.38
3	100	.53

Fitness = (.42 + .17 + .38 + .53) / 4

= .375

Figure 50. The components of a simple NK landscape where $N = 4$ and $K = 2$. The table on the left shows the fitness contribution (rounded to two decimal places) for each locus for each possible neighborhood configuration. The small diagram at the top right shows the loci that influence each locus. An illustration of the fitness calculation for the vertex 0110 is shown in the bottom right.

this chapter investigate the relative performance of the various hillclimbing algorithms on a set of peaks. In these experiments, the aim is to provide an answer to the question: Which hillclimbing algorithm can be expected to locate a peak using the fewest function evaluations? The experiment considers the randomly located peaks and also sets of high-fitness peaks. On the studied instances of the busy beaver problem, the optimal peaks are all known, and the performance of the hillclimbers is compared given the task of locating any one of the optimal peaks. For the NK landscapes, the best five percent of the randomly located peaks are separated, and the hillclimbers are compared when their task is to find any one of these peaks. The reverse hillclimbing algorithm is used to make predictions about the performance of the four hillclimbers and experiments show these to be very accurate.

The reverse hillclimbing analysis helps to throw light on an important factor in hillclimbing algorithms. This is the question of how much time should be spent looking for uphill neighbors before moving. Reverse hillclimbing makes it possible to see that, on the problems considered, hillclimbers that choose higher neighbors have higher probabilities of reaching higher peaks. Choosing a higher neighbor incurs a cost however. The hillclimber must examine more neighbors. Finding a good balance between these two is important to overall performance. AA and SA represent quite different responses to this tradeoff. AA performs few function evaluations to find any uphill neighbor whereas SA examines all neighbors to guarantee that one with maximal fitness is selected. Several new hillclimbers that explore this tradeoff are examined in Appendix B.

In the experiments of this chapter, three NK landscapes were investigated. In each of these, the value of N was 16. K took on values of 4, 8 and 12. The 2-, 3- and 4-state busy beaver problems are examined.

4.9. Results

4.9.1. NK Landscapes

For each of the three NK landscapes examined, a number of peaks were identified (via hillclimbing). These were then subject to reverse hillclimbing using each of the four hillclimbing algorithms. Then, for each of these landscapes, the fittest five percent of the peaks were examined separately to see if one of the algorithms appeared better at finding higher peaks.

Table 7 shows the result of reverse hillclimbs from 1000 peaks on an $N = 16$, $K = 12$ landscape. The peaks were located by performing hillclimbs from randomly chosen starting points. The four columns of the table represent the four algorithms. The first row of data shows the number of different basin sizes that were found. For example, of the 1000 different peaks whose basins were explored for LA, there were only 176 different basin sizes found. The second, third, fourth and fifth lines show the minimum, maximum, mean and standard deviation of the basin sizes that were found. The second last line shows the overall probability that a hillclimb started from a randomly chosen location will reach one of the peaks from the set of 1000. The final row is the expected number of climbs that it would take for this to occur. This row is the reciprocal of the previous row.

Table 7. Reverse hillclimbing from 1000 random peaks in a 16,12 NK landscape.

	AA	LA	MA	SA
# of basin sizes	962	176	413	150
Min. basin size	7268	1	1	2
Max. basin size	35,841	541	1230	287
Mean basin size	25,634	35	178	47
Basin size s.d.	4584	62	154	38
P(Discovery)	0·645	0·534	0·619	0·713
E(Climbs)	1·548	1·873	1·617	1·402

There are several things that can be noted about this table:

- AA has basins that are far greater in size than those of any other algorithm. The smallest basin found under AA is approximately six times bigger than the maximum basin size under any other algorithm. That basins under AA are bigger than the basins under the other algorithms is not surprising. If one orders the hillclimbers in terms of decreasing out-degree of landscape vertices, the ranking would be AA first, MA second and LA and SA roughly equal last. The same ranking can be observed in mean (and maximum) basin size. Note that the ranking of algorithms by landscape out-degree would change if N were odd instead of even, since in that case MA would find a single median instead of having to make a random choice between two at every step of its climb.

- AA also produces far more unique basin sizes, which is not surprising given the far greater range of these numbers. Of the 1000 basins that were explored, there were 962 different basin sizes found. What seems initially more remarkable is the low numbers for the other algorithms, especially for LA and SA. This proves to have an uninteresting explanation however, which is that there are many small basins for each of these algorithms. If a hillclimber tends to have small basins, there will tend to be fewer different basin sizes in a sample. This is confirmed by looking at the actual basin sizes (not shown). Further evidence that this is the cause is seen by noting the agreement between the maximum basin sizes and the number of different basin sizes. If we rank the four algorithms by decreasing maximum basin size, and also by decreasing number of basin sizes, we get identical orderings.

- The 1 entries for minimum basin size under LA and MA are eye-catching. If each of the vertices from which reverse hillclimbing is performed in this experiment is supposed to be a peak, how can any of them have a basin size of only one (i.e., the peak and no other vertices)? This seemed at first suspicious but has a simple solution. These vertices are indeed peaks, which is to say that all their neighbors are less fit than they are. But their basins also have size one, because each of the neighbors does not ascend to the peak. This cannot happen under AA since there is always a non-zero probability of an ascent to every fitter neighbor. It is possible in the other three algorithms. It did not happen with SA in this set of peaks, but that will be seen in later experiments.

  These peaks are something of a curiousity, and are without a doubt the hardest peaks to locate with these algorithms, since their basin of attraction contains only the peaks themselves. To find such a peak with a randomly started hillclimb, the peak itself must be chosen as the starting vertex. As a result, LA, MA and SA cannot do better than random search when attempting to locate such a vertex.

- The overall probability of locating a peak shows an interesting pattern. If we ignore AA for the time being, there is a direct correspondence between how exploitative LA, MA and SA are and how often they expect to find one of the set of peaks. In saying that one algorithm is more exploitative than another, it is meant that if all the fitter neighbors of a vertex were discovered and ranked from low to high according to fitness, the more exploitative algorithm would choose to move to one whose rank was higher. SA is more exploitative than MA which in turn is more exploitative than LA. Correspondingly, SA has a higher probability of finding one of the set of peaks than MA which has a higher probability than LA.

Since AA is on average as exploitative as MA (though with far greater variance), it seems reasonable to expect that its probability of discovery lies between that of SA and LA, and closer to MA than to either of the extremes. This is in fact the case. These general patterns of discovery probabilities will persist throughout the experiments in this chapter. At this stage, it should be remembered that these results apply to a set of peaks that were located via randomly started hillclimbs. We have not seen if the pattern persists when we ask what the probabilities are of locating good peaks.

As the 1000 peaks whose basins were calculated were found via randomly started hillclimbs, there is nothing special about them and the fact that one algorithm can find one of them with a higher probability than another is not particularly interesting. More interesting is to consider the best of these peaks to see if any algorithm appears to have an advantage at finding higher peaks. There does not seem to be any a priori reason to expect such a bias, but it is worth looking for. If this bias is not found, it is a good indication that the performance of the hillclimbing algorithms may be accurately estimated via observing their performance on peaks located through random sampling.

Table 8 is identical in form to Table 7, except the figures displayed show only the data for the best five percent (50) of the original 1000 peaks. Points of interest in Table 8 include the following:

- The number of different basin sizes has leveled out across the algorithms. This is to be expected, since we are now sampling the fittest peaks, and it is not unreasonable to expect these to have bigger basins and thus, probably more variance in their basin sizes.

- Interestingly, the 1 entries for LA and MA have persisted, when they might have been expected to disappear as it seems reasonable to expect that they would correspond to particularly poor peaks. This is definitely not the case. Further investigation reveals that 9 of the top 50 peaks have basin sizes of 1 under LA. The neighbors of a very fit vertex, v, are likely to have other neighbors that are less fit than v, and in this case, LA will not move to v from any of its neighbors. MA exhibits the same qualitative behavior, but its effect is greatly reduced (only 1 of the top 50 peaks had a basin of size 1 under MA). Under LA, 15 of the top 50 peaks had a basin of size less than 5, whereas under MA

Table 8. Reverse hillclimbing from the best 50 of 1000 random peaks in a 16,12 NK landscape.

	AA	LA	MA	SA
# of basin sizes	50	30	48	43
Min. basin size	19,242	1	1	28
Max. basin size	35,130	385	1230	287
Mean basin size	29,243	45	313	102
Basin size s.d.	3489	67	241	59
P(Discovery)	0·062	0·034	0·053	0·078
E(Climbs)	16·18	29·39	18·80	12·88

only 2 did. It is not hard to see that the global maximum of these function will always be such a peak under LA and MA! Clearly, LA and MA will not be much help in locating this vertex.

- The pattern of ascent probabilities described above has also persisted. Again, AA comes out slightly ahead of MA.

This information allows us to answer the question of which hillclimber to bet on if we are allowed some number of climbs. But as we are concerned with evaluations, not runs, it would be good to answer the question: Which of the hillclimbing algorithms appears to be the best, in terms of evaluations, at finding this set of peaks? The data from the reverse hillclimbing is exact, but is not enough. If we consider the iterated version of each of these algorithms, a successful search for one of the set of desired peaks will consist of some number of unsuccessful climbs (that do not reach a peak in the desired set) followed by a successful climb. If a problem is at all difficult, there will be many unsuccessful climbs before a peak is found. If a reasonably accurate estimate of the number of evaluations taken by an algorithm on a climb were available, it could be combined with the exact reverse hillclimbing data to obtain an equally accurate prediction of the number of evaluations needed by that algorithm to locate a peak.

Information about the average number of evaluations used in a climb (successful or not) is easily obtained. Table 9 shows a summary of 10,000 climbs on the $N = 16$, $K = 12$ landscape for the four algorithms. As would be expected, AA uses significantly fewer evaluations per climb than the other algorithms, all of which examine all neighbors before moving. The reason LA, MA and SA do not perform similar numbers of evaluations is that they go uphill at different rates, as can be seen from the mean steps per climb in the table. SA has a mean of about 2 steps before reaching a peak, while LA takes about 9.

Table 9. The result of 10,000 hillclimbs on the 16,12 NK landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	3·455	9·335	3·836	1·999
Min. steps/climb	0	0	0	0
Max. steps/climb	13	33	12	8
Steps/climb s.d.	1·878	5·919	1·857	1·064
Av. evals/climb	31·22	166·4	78·37	48·99
Min. evals/climb	17	17	17	17
Max. evals/climb	90	545	209	145
Evals/climb s.d.	10·63	94·70	29·66	17·02

The most interesting question that arises is whether AA will be better than SA or vice-versa. SA has a lower expected number of climbs to locate a peak, but takes more evaluations to do so. The answer to this can be found by multiplying the expected number of climbs from Table 8 by the expected number of evaluations from Table 9. This produces Table 10.

The table compares the expected performance of each of the hillclimbers with their observed performance. SA located more peaks in the 10,000 runs but, more importantly, AA took fewer evaluations on average to locate a peak. Since we are using algorithm evaluation count as our yardstick for algorithm performance, AA is

Table 10. Expected and observed numbers of peaks and evaluations for 50 good NK 16,12 peaks.

	AA	LA	MA	SA
E(Peaks found)	617	340	532	777
Peaks found	620	348	578	736
E(Evals/peak)	505	4889	1473	631
Evals/peak	503	4780	1355	665

the best algorithm for finding this particular set of peaks on this particular landscape, assuming the sampling that gave the expected evaluations per climb data was not wildly wrong. SA is a clear second, with MA and LA third and fourth. These results could easily be made more accurate by increasing the number of hillclimbs done to obtain the data on expected number of evaluations per climb. For our purposes, a small number of hillclimbs to estimate evaluations per climb was sufficient to obtain estimates that closely matched the actual performance of the algorithms.

Of course, the behavior of the different algorithms may simply be due to chance, and may not apply to other NK landscapes, let alone other problems. To investigate this, and also to examine landscapes of differing ruggedness, the same experiment was performed on an $N = 16$, $K = 8$ and an $N = 16$, $K = 4$ NK landscape. Tables 11 to 14 are the $N = 16$, $K = 8$ counterparts of Tables 7 to 10 for the $N = 16$, $K = 12$ landscape. Table 11 shows the result of performing reverse hillclimbing from 700 peaks in a $N = 16$, $K = 8$ landscape. Less peaks are descended from in this experiment as the landscape contains fewer peaks. Judging from the figures for SA in the table, the 700 peaks and their basins account for approximately 96% of the vertices in the landscape. This follows from the fact that SA is with high probability completely deterministic on an NK landscape (since it is very unlikely that any two vertices will have identical fitness), so multiplying the mean basin size (90) by the number of basins (700) gives a good estimate of the number of vertices represented by the sample. In this case the basins represent approximately 63,000 of the 2^{16} vertices. The 2500 or so vertices that are not represented should (if the mean basin size can be used reliably) fall into approximately 28 basins. We would expect only one of these to have fitness in the highest 5% of all basins, so we can be reasonably confident that

Table 11. Reverse hillclimbing from 700 random peaks in a 16,8 NK landscape.

	AA	LA	MA	SA
# of basin sizes	687	233	526	230
Min. basin size	11,179	1	1	1
Max. basin size	49,275	975	4399	973
Mean basin size	34,812	88	583	90
Basin size s.d.	6613	145	659	109
P(Discovery)	0·958	0·945	0·958	0·966
E(Climbs)	1·044	1·059	1·044	1·035

the top 5% of our sample of 700 is very close to being the top 5% of peaks in the entire landscape.

The patterns that were evident in the $N = 16$, $K = 12$ landscape can also be seen on this landscape. Once again, SA has a higher probability of locating a peak than MA does and it in turn has a higher probability than LA. AA has a probability that is very close to that of MA (in the table they are identical as only two decimal places are shown). The basin sizes also show patterns that are the same as those on the more rugged landscape. AA has by far the biggest basins, then MA and then SA just barely ahead of LA. Table 12 shows the reverse hillclimbing results when we restrict attention to the top 5% (35) of the 700 peaks. Again, looking at just the best peaks in the sample does not alter the relative algorithm performances.

Table 13 shows the result of 10,000 hillclimbs on this landscape. Once again it is clear that one of SA and AA will prove to be the best algorithm on this landscape. SA has a better chance of finding a peak on a given climb, but AA takes fewer comparisons per climb. The expected and observed data for the number of peaks discovered and the number of evaluations per discovery is presented in Table 14 and the balance once more favors AA. SA's dominance in probability of discovery does not match AA's dominance in evaluations per climb.

This story is repeated in every detail on the $N = 16$, $K = 4$ landscape. In this case, all 180 peaks in the landscape were identified. The reverse hillclimbs from

Table 12. Reverse hillclimbing from the best 35 of 700 random peaks in a 16,8 NK landscape.

	AA	LA	MA	SA
# of basin sizes	34	31	35	32
Min. basin size	35,751	1	618	73
Max. basin size	48,486	975	14,399	829
Mean basin size	43,704	191	2019	349
Basin size s.d.	3393	236	1044	145
P(Discovery)	0·159	0·102	0·156	0·186
E(Climbs)	6·294	9·809	6·411	5·372

these peaks are shown in Table 15. The best 5% (9) are examined in Table 16, 10,000 hillclimbs were done to gather expected uphill path lengths (Table 17), and the summary results appear in Table 18.

In this case, the basins of attraction of the peaks account for all the vertices in the landscape. As we have seen virtually identical behavior on these three landscapes, it seems reasonable to assume that although we only considered about 70% of the space in the $N = 16$, $K = 12$ landscape, that the results would not have been greatly different had we examined additional peaks.

4.9.2. Busy Beavers

Table 19 shows the result of performing reverse hillclimbing on 1000 randomly located peaks on the 4-state busy beaver landscape. As with the NK landscape experiments above, the peaks are found via hillclimbs started from randomly chosen starting points. With the NK landscapes, the union of the points in all the basins of attraction of the set of randomly chosen peaks comprised a large proportion of the space. For example, with SA, over 70% of the space was in the basin of attraction of the 1000 peaks chosen on the $N = 16$, $K = 12$ landscape (see Table 7 on page 101). This is certainly not the case with the 1000 randomly located peaks in the 4-state busy beaver problem. If we (generously) assume that basins do not overlap, the union of

Table 13. The result of 10,000 hillclimbs on the 16,8 NK landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	4·632	12·62	5·092	2·833
Min. steps/climb	0	0	0	0
Max. steps/climb	14	38	13	10
Steps/climb s.d.	2·257	7·062	2·244	1·317
Av. evals/climb	36·37	218·8	98·47	62·33
Min. evals/climb	17	17	17	17
Max. evals/climb	103	625	225	177
Evals/climb s.d.	12·50	112·9	35·91	21·07

points in the basins of all the peaks under SA is a set of 777,000 points. In an overall space of 25,600,000,000 points, this represents only just over 3×10^{-5} of the entire space.

Table 20 shows the result of reverse hillclimbing from the 48 known optimal TMs for the 4-state problem. The small number of different basin sizes for the algorithms is the result of redundancies in the representation. Once again, the four algorithms exhibit an ordering, by probability of reaching any peak on a single hillclimb, that should by now be becoming familiar. SA is a clear winner, AA is second, followed by MA and then LA. This pattern was also seen in the best five percent of peaks for the three NK landscapes (see Tables 8 to 16). In all three of those cases, SA ranked first and AA second. In two of them, MA was third and LA last and in one, LA third and MA last. This pattern will be seen in the three busy beaver landscapes. In fact, the pattern in the busy beaver landscapes is *exactly* that seen in the NK landscapes. SA is first in all three, AA is second in all three and MA beats LA on all but the simplest of the problems.

To examine whether the reverse hillclimbing results are accurate and to determine which algorithm can be expected to find peaks using the fewest evaluations, it is

Table 14. Expected and observed numbers of peaks and evaluations for 35 good NK 16,8 peaks.

	AA	LA	MA	SA
E(Peaks found)	1589	1019	1560	1862
Peaks found	1451	1042	1458	1854
E(Evals/peak)	229	2146	631	335
Evals/peak	251	2100	675	336

necessary to estimate the expected number of evaluations in a hillclimb, as was done in the NK landscape experiments. Table 21 shows the statistics obtained from 10,000 hillclimbs for each of the algorithms. Again we see a similar pattern to that seen for the NK problems. AA has the fewest expected evaluations and is followed by SA, MA and LA in that order. This relationship has a simple explanation, that was presented in §4.9.1(105). The most striking aspect of this table is the very small number of steps that a hillclimber can expect to take on this landscape before reaching a peak. For SA, the expectation is less than 2 steps.

Table 22 compares the actual number of peaks found and evaluations taken with the expected numbers. Because it is very difficult to locate these peaks with hillclimbing, each algorithm was run 20 million times to obtain a reasonable number of successes. The expected and observed numbers of peaks are in good agreement, as are the numbers of evaluations. In this case, the advantage that SA has over AA in terms of its probability of finding a peak outweighs its disadvantage in terms of expected number of evaluations per climb. This is the first time we have have seen the tradeoff resolved in favor of SA, and the victory is a very slight one. If we assume that the estimated number of evaluations per climb is actually correct, then the difference in the expected number of evaluations per peak location between the two algorithms is insignificant.

This experiment was repeated for peaks on the 3- and 2-state busy beaver problems. These results are shown in Tables 23 to 30. In both of these problems, SA again has the highest probability of peak location given a single climb, but is beaten by AA due to the high cost it incurs per uphill step.

Table 15. Reverse hillclimbing from the 180 peaks in a 16,4 NK landscape.

	AA	LA	MA	SA
# of basin sizes	179	140	178	158
Min. basin size	14,703	1	72	6
Max. basin size	55,146	5029	14,207	2565
Mean basin size	43,640	364	2761	364
Basin size s.d.	7944	588	2738	446
P(Discovery)	1·0	1·0	1·0	1·0
E(Climbs)	1	1	1	1

Table 16. Reverse hillclimbing from the best 9 of the 180 peaks in a 16,4 NK landscape.

	AA	LA	MA	SA
# of basin sizes	9	9	9	9
Min. basin size	42,817	69	4566	644
Max. basin size	54,927	5029	14,207	2485
Mean basin size	51,768	1161	7637	1444
Basin size s.d.	3723	1470	3499	634
P(Discovery)	0·165	0·160	0·151	0·198
E(Climbs)	6·076	6·269	6·618	5·042

Table 17. The result of 10,000 hillclimbs on the 16,4 NK landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	6·064	15·20	6·622	3·886
Min. steps/climb	0	0	0	0
Max. steps/climb	19	44	19	10
Steps/climb s.d.	2·587	3·373	2·695	1·484
Av. evals/climb	41·58	260·2	122·9	79·18
Min. evals/climb	17	17	17	17
Max. evals/climb	126	721	321	177
Evals/climb s.d.	13·57	117·9	43·12	23·74

Table 18. Expected and observed numbers of peaks and evaluations for 9 good NK 16,4 peaks.

	AA	LA	MA	SA
E(Peaks found)	1645	1595	1511	1983
Peaks found	1530	1592	1545	2017
E(Evals/peak)	252	1631	813	399
Evals/peak	272	1634	795	393

Table 19. Reverse hillclimbing from 1000 random 4-state busy beaver peaks.

	AA	LA	MA	SA
# of basin sizes	470	440	320	356
Min. basin size	4	4	1	2
Max. basin size	545,848	360,864	42,970	61,606
Mean basin size	4305	2585	481	777
Basin size s.d.	27,647	16,521	2628	4282
P(Discovery)	0·0000075	0·0000066	0·0000071	0·0000085
E(Climbs)	133,754	151,713	141,453	117,993

4.10. Conclusion

This chapter introduced the reverse hillclimbing algorithm. This chapter showed that the algorithm makes it possible to determine the relative performance of two hillclimbing algorithms (that only take uphill steps) on a given problem. The algorithm also provides answers to several other questions concerning sizes of basins of attraction and lengths of paths to discover peaks. To answer these questions without reverse hillclimbing, it appears necessary to use statistical methods based on the occasional discovery of peaks. These methods are both computationally infeasible and of limited accuracy. Reverse hillclimbing, when it is useful, provides answers to the above questions very rapidly and with a high degree of precision. In many cases answers obtained with the method are exact.

The reverse hillclimbing algorithm cannot be used to discover high peaks on a fitness landscape. Rather, it is used as the basis for analysis once these peaks have been found by other methods. For instance, if a certain hillclimbing method has identified 100 peaks on a "rugged" landscape, reverse hillclimbing can be used to calculate the probability that other hillclimbers would find these peaks. If some other search algorithm has identified a good peak, reverse hillclimbing can produce the exact probability that a given form of hillclimbing would have found the point. This allows rapid and accurate comparisons without the need to run the hillclimber

Table 20. Reverse hillclimbing from the 48 peaks in the 4-state busy beaver problem.

	AA	LA	MA	SA
# of basin sizes	6	5	24	4
Min. basin size	10,420	3349	897	1426
Max. basin size	25,714	5648	2660	5364
Mean basin size	18,072	4511	1552	3370
Basin size s.d.	7637	1133	669	1936
P(Discovery)	0·000001169	0·000000610	0·000001002	0·000002012
E(Climbs)	855,538	1,639,043	997,524	497,129

to form an estimate of the probability. In previous work [110], this approach was used to assess the performance of a GA.

Reverse hillclimbing has the pleasing property of becoming more feasible as the landscape on which it is run gets more "rugged." That is, as basin sizes decrease (perhaps due to increased numbers of peaks, as we saw above with NK landscapes) reverse hillclimbing becomes increasingly competitive with sampling methods based on hillclimbing. On extremely difficult problems, reverse hillclimbing will perform extremely well since it never strays more than one vertex from the points in the basin of attraction during its descent. Therefore, the smaller the basin of attraction of a peak, the faster reverse hillclimbing will be and the more difficult it becomes to answer questions about the basin using other methods.

Other statistics about landscapes can also be easily investigated via reverse hillclimbing. These include the relationships between peak heights, basin sizes and probability of discovery. A trivial modification to the algorithm can be used to explore the "basin" of points *above* a certain point in a landscape. This allows calculation of statistics relating the number of uphill directions to the number of peaks that can be reached from a given point. This information might be used to direct the behavior of a hillclimber that expended variable effort in looking for uphill neighbors at each stage of the climb.

Table 21. The result of 10,000 hillclimbs on the 4-state busy beaver landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	2·091	2·242	2·111	1·883
Min. steps/climb	0	0	0	0
Max. steps/climb	8	9	8	8
Steps/climb s.d.	1·316	1·461	1·307	1·165
Av. evals/climb	83·35	156·6	150·3	139·4
Min. evals/climb	49	49	49	49
Max. evals/climb	200	481	433	433
Evals/climb s.d.	25·82	70·14	62·73	55·92

Finally, reverse hillclimbing has highlighted the tradeoff between moving uphill quickly and moving in the steepest direction possible. The any ascent algorithm takes the first uphill direction it can find and consequently expends little energy looking for uphill directions. Steepest ascent always examines all neighbors and then chooses the steepest of these. Although, in our experience, steepest ascent always has a higher probability of finding a peak on a single climb (this can be shown exactly by reverse hillclimbing for a given problem), this advantage is usually not enough to compensate for the extra work it must do at each step. These two approaches represent extremes of behavior. Appendix B presents preliminary results of an attempt to find more balanced hillclimbers.

Table 22. Expected and observed numbers of peaks and evaluations for the 48 peaks on the 4-state busy beaver problem. The observed values are taken from 20 million hillclimbs with each of the four algorithms.

	AA	LA	MA	SA
E(Peaks found)	23	12	20	40
Peaks found	23	15	23	44
E(Evals/peak)	71,307,723	256,657,743	149,960,575	69,294,214
Evals/peak	71,270,514	208,393,004	130,625,970	63,161,171

Table 23. Reverse hillclimbing from 1000 random 3-state busy beaver peaks.

	AA	LA	MA	SA
# of basin sizes	287	279	212	224
Min. basin size	4	4	2	2
Max. basin size	20,219	13,616	7052	8142
Mean basin size	327	222	108	137
Basin size s.d.	1309	869	409	502
P(Discovery)	0·00368	0·00329	0·00364	0·00407
E(Climbs)	272	304	275	246

Table 24. Reverse hillclimbing from the 40 peaks in the 3-state busy beaver problem.

	AA	LA	MA	SA
# of basin sizes	13	14	22	14
Min. basin size	5659	2522	1247	2092
Max. basin size	20,219	13,616	7052	8142
Mean basin size	10,974	6702	3249	4156
Basin size s.d.	6013	4746	2183	2426
P(Discovery)	0·00399	0·00337	0·00381	0·00453
E(Climbs)	251	297	262	221

Table 25. The result of 10,000 hillclimbs on the 3-state busy beaver landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	1·754	1·829	1·762	1·591
Min. steps/climb	0	0	0	0
Max. steps/climb	7	7	7	6
Steps/climb s.d.	1·114	1·189	1·116	1·005
Av. evals/climb	50·65	85·87	83·86	78·72
Min. evals/climb	31	31	31	31
Max. evals/climb	136	241	241	211
Evals/climb s.d.	15·64	35·68	33·47	30·14

Table 26. Expected and observed numbers of peaks and evaluations for the 48 peaks on the 3-state busy beaver problem. The observed values are taken from 10,000 hillclimbs with each of the four algorithms.

	AA	LA	MA	SA
E(Peaks found)	40	34	38	45
Peaks found	49	29	37	48
E(Evals/peak)	12,687	25,463	21,990	17,367
Evals/peak	12,662	29,609	22,665	16,400

Table 27. Reverse hillclimbing from 1000 random 2-state busy beaver peaks.

	AA	LA	MA	SA
# of basin sizes	37	35	27	27
Min. basin size	4	4	4	3
Max. basin size	372	292	200	274
Mean basin size	14·4	12·1	11·0	11·3
Basin size s.d.	26·4	20·7	14·2	19·0
P(Discovery)	0·409	0·393	0·406	0·425
E(Climbs)	2·448	2·545	2·460	2·355

Table 28. Reverse hillclimbing from the 4 peaks in the 2-state busy beaver problem.

	AA	LA	MA	SA
# of basin sizes	2	2	2	2
Min. basin size	370	290	198	273
Max. basin size	372	292	200	274
Mean basin size	371	291	199	273·5
Basin size s.d.	1	1	1	0·5
P(Discovery)	0·0354	0·0321	0·0304	0·0390
E(Climbs)	28·27	31·17	32·85	25·64

Table 29. The result of 10,000 hillclimbs on the 2-state busy beaver landscape. The table shows data for the number of uphill steps taken for each algorithm and the number of evaluations performed per uphill walk.

	AA	LA	MA	SA
Av. steps/climb	1·372	1·446	1·374	1·305
Min. steps/climb	0	0	0	0
Max. steps/climb	5	5	5	5
Steps/climb s.d.	0·917	0·980	0·911	0·838
Av. evals/climb	26·26	40·14	38·99	37·87
Min. evals/climb	17	17	17	17
Max. evals/climb	76	97	97	97
Evals/climb s.d.	7·898	15·68	14·57	13·41

Table 30. Expected and observed numbers of peaks and evaluations for the 48 peaks on the 2-state busy beaver problem.

	AA	LA	MA	SA
E(Peaks found)	354	321	304	390
Peaks found	356	317	290	364
E(Evals/peak)	742	1251	1281	971
Evals/peak	737	1266	1344	1040

CHAPTER 5

Evolutionary Algorithms and Heuristic Search

5.1. Introduction

Although evolutionary algorithms are often considered a technique of AI, the classification is more due to convenience than to any strong claims of connections between the fields. This chapter establishes such a connection. In this chapter, the framework of a correspondence between evolutionary algorithms and heuristic state space search is presented, and one aspect of the correspondence is developed in detail.

The view of evolutionary algorithms by researchers in the more established fields of AI is colored to a large extent by uniform use of biological language. Putting that language aside temporarily allows the development of a perspective on these algorithms from which they appear to be no more than slightly unusual state space search algorithms. Much of this shift in perspective can be accomplished by simply using slightly different language to describe these algorithms. Having accomplished this, it is possible to put the correspondence to work by asking whether aspects of research in one field can be usefully translated into the other. This possibility was also raised by Tackett [119, 120] in the context of genetic programming.

Some classifications of approaches to AI made at a recent workshop will serve to illustrate the differences in how evolutionary and state space search are typically viewed [121]:

- As categories of approaches to AI, state space search was classified as a *Symbol Processing* while evolutionary algorithms, represented by genetic programming, were classified as *Biocomputational.*

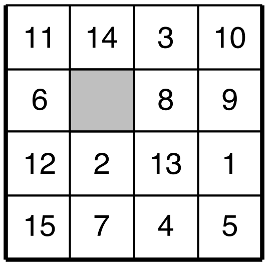
- In another classification (proposed by Nilsson), state space search was classified as both "engineering (computer science, control theory,...)" and "top down," whereas evolutionary algorithms were "biological science (neuroscience, ethology, evolution,...)" and "bottom-up."

- In a final classification (proposed by Rumelhart), state space search was both "high-level" and "formal" whereas evolutionary algorithms were "low-level" and "experimental."

This chapter demonstrates that a quite different classification is possible and that it can be used to inform the study of GAs.

5.2. Search in Artificial Intelligence

AI has long been concerned with questions of search. Search forms the trunk of the tree from which many AI techniques have sprouted. Minsky [25] divided the problems of AI into five areas: Search, Pattern Recognition, Learning, Planning and Induction. Of which pattern recognition, learning, and planning are mentioned as direct responses to the general problem of search. The most fundamental view of search in AI is that of a search to locate an object within a space of possible objects. This leads directly to the idea of *state space search*, in which objects are regarded as states that the searcher examines in an attempt to locate a solution. General descriptions of this sort can be found, for example, in [31, 43, 45, 122]. The "15-puzzle" [123] depicted in Figure 51 is commonly used to illustrate this view. One can imagine the entire set of possible arrangements (states) in a connected fashion such as that shown in Figure 52. Puzzle configurations are connected through the action of sliding a tile into the unoccupied location. This is known as a state space and the problem can be viewed as that of finding a path through this network to the unique solution state.



11	14	3	10
6		8	9
12	2	13	1
15	7	4	5

Figure 51. The 15-puzzle. The object is to rearrange the fifteen numbered tiles to form an increasing sequence from top left to bottom right. Tiles are rearranged via the movement of a numbered tile into the vacant (dark) location.

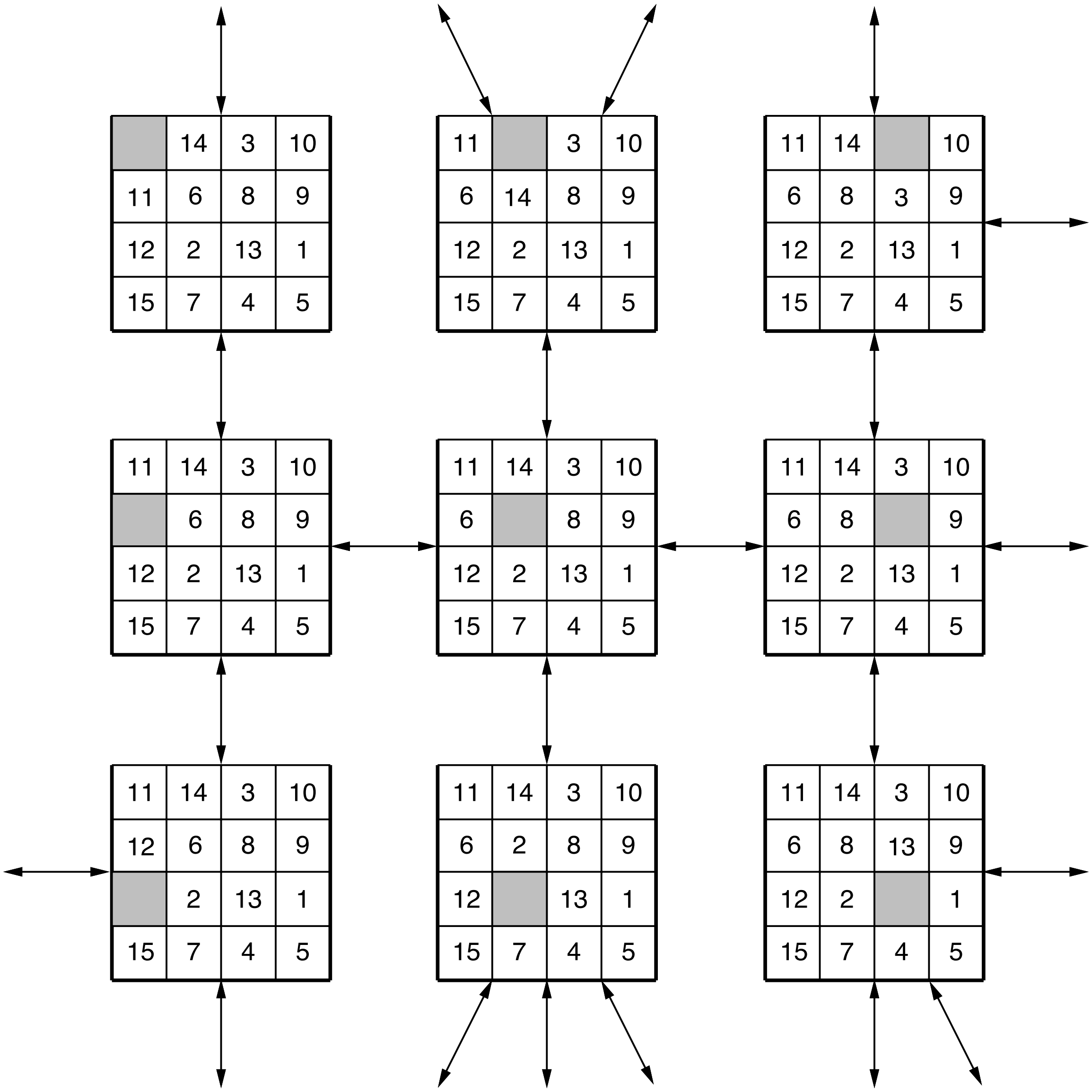


Figure 52. A portion of the state space for the 15-puzzle. Sliding a tile into the unoccupied location transforms one state into another.

Viewing a problem as a search within a state space is very common in AI. A state space and the operations used to move between states produces a directed graph. This is the connection with the landscape model of this dissertation and is the subject of this chapter. AI has developed many algorithms that are designed to search graphs. In these algorithms, the nodes of the graph are labeled with the value of some heuristic function that provides an estimate of the worth of the state, often in terms of its distance from a goal state. State space graphs are also known within AI as OR-graphs in contrast to the more general AND/OR-graphs [124]. Pearl demonstrates that it is sometimes possible to adopt either view when solving a problem and that the choice between them can have a large influence on the difficulty of finding a solution [31]. This chapter is not concerned with AND/OR-graphs, and seeks only to establish connections between landscapes and state spaces.

5.3. Similarities Between Evolutionary Algorithms and Heuristic Search

The correspondence between these two classes of algorithms is based on the fact that both can be seen as searching labeled graphs. This section presents various aspects of the correspondence in enough detail to show that the resemblance is not superficial. Table 31 summarizes the different language used to describe essentially the same things in the two fields. The correspondence between fitness functions and heuristic functions is investigated in detail in the second half of this chapter.

5.3.1. Landscapes and State Spaces

Figure 52 shows that a state space may be naturally viewed as a directed graph. If we consider the origin of a state space in the same way that we considered the origin of the components of a landscape in §2.3(19), important parallels can be observed.

Considering a problem as a state space search is so common within AI. This step may appear so natural, that it is easy to overlook the fact that the searcher makes a decision to do so, and thus must make a choice about how this should be done. When introducing state space search, many AI texts (for example [125, 126, 127, 128, 129, 130, 131, 132, 133, 134]), adopt a particular state space view of a problem without explicit acknowledgment of the fact that doing so is a choice or that other choices of state space are possible. This is generally because in the examples used, there is an obvious and natural state space that it is simple to adopt. This choice of state space

Table 31. The languages of evolutionary and heuristic search algorithms.

Evolutionary algorithms	Heuristic state space search
Landscape	State space (or database)
Individual	Potential Solution
Navigation strategy	Control strategy
Fitness function	Heuristic function
Global Optima	Goal states
Population	OPEN list (for example)
Operator	Operator

is also often recommended by the structure of the problem at hand. Exceptions to this trend can be found in [43, 44, 135].

But the statement of a search problem usually says *nothing* about states. If one thinks of many of the classic search problems that AI has dealt with, for example the n-queens problem, the 8-puzzle, the 15-puzzle, Rubik's Cube, Nim, Tic-Tac-Toe, sliding tile puzzles, missionaries and cannibals, and the Tower of Hanoi, in no case does the problem statement say anything about states or state spaces. The state space is constructed by the problem solver, and is therefore a choice that is made when solving a problem.

The move to considering a problem from a state space point of view can be so natural that it is easy to overlook. Emphasizing the fact that a choice is being made leads to questions about alternatives. In order to adopt a state space point of view, we must first decide what it is that constitutes a state. Often in AI problems there is an underlying structure which is imposed by physical properties of the problem (e.g., manipulating Rubik's Cube) or by rules of a game. Such a structure, which I call *predefined connectivity* produces a *natural state space.* Examples of problems with predefined connectivity include all those given above with the exception of the n-queens problem. The existence of predefined connectivity does not preclude the construction of state spaces other than the natural one. In this case, states will consist of sets of states from the natural state space and operators will be compositions of operators in the natural state space. Rubik's Cube is useful for illustrating this choice.

The natural state space representation for the cube has each configuration of the cube correspond to a state in the state space graph, with edges emanating from a vertex given by the possible twists of the cube's sides. A second choice is to construct a state space with only three elements, one being the set of all cube configurations that have one layer complete, the second the set of all cube configurations with two layers complete and the third the completed cube. This state space would be very easy to search if we could find the operators that connected the states. In a third view, used by Korf [33], the individual "cubies" and the possible cubie positions are labeled from 1 to 20. State $i.j$ (for $1 \leq i \leq 20$ and $i < j \leq 20$) indicates that cubies numbered less than or equal to i are all positioned correctly and that cubie $i+1$ is in position j. When $i = 20$ the puzzle is solved and the value of j is irrelevant. In this state space, each state corresponds to a set of cube configurations. This state space also has the advantage of being trivial to navigate in. Fortunately, it is possible (though by no means easy) to find operators that connect successive states [33]. Korf calls the sequences of cube twists that make up the operators in this state space *macro-operators* and attributes the idea to Amarel [32]. The problem of finding such operators is a challenging search problem in itself. Korf's preference is to continue to regard the natural state space as the state space under investigation, hence the name macro-operator. Mine is to treat sets of states of the natural state space as states in a new state space and the macro-operators as operators in the new state space. The relation "in the same Korf $i.j$ class as" is an equivalence relation over the set of all cube configurations. Each of the equivalences classes defined by this relation can be regarded as a state in a new state space. Allowing a state in a state space to correspond to a set of states from another state space is merely an example of allowing a landscape vertex to correspond to a multiset of elements of $\mathcal{R}$.

These choices are all possible approaches to solving Rubik's cube by viewing the search as taking place in a state space. The first state space is simple to construct and hard to search and the second and third are hard to construct and simple to search. This tradeoff is exactly the tradeoff that we encountered in §2.11(41) when discussing operators and representation in the context of landscapes. The issues that we face when constructing a state space and operators to move between states are the same as those that we face when choosing an object space (and, subsequently, a representation space) and operators when constructing a landscape. In both cases we need to decide what we will focus on as possible solutions to the problem and how we will transform these possible solutions into others. In both cases, the results of these choices can be viewed as a graph in which the search will take place.

5.3.2. Individuals, Potential Solutions, and Populations

Evolutionary algorithms usually deal with populations of individuals. A vertex in a landscape graph corresponds to a multiset of individuals drawn from $\mathcal{R}$. In some landscapes, these multisets will always contain a single individual (e.g., in mutation) while in others they will correspond to pairs of individuals (crossover) or entire populations (as in selection and in the view of an evolutionary algorithms as a Markov chains). In state spaces we have a very similar situation. A state in a state space might correspond to a single configuration of the 15-puzzle (which we could call an individual if we pleased), or perhaps to some piece of a Hamiltonian circuit in a graph. This can be thought of as a partial individual, but also as representing the set of all Hamiltonian circuits that contain this piece as a subgraph. This set may be thought of as a population. In many AI search algorithms, a list of potential solutions (often called the OPEN list) is kept. The contents of this list is very much like a population and the search algorithm chooses promising directions (individuals) to explore from. This point was also made by Tackett [119, 120] in the context of a correspondence he develops between genetic programming and beam search. Whether we call the objects that correspond to vertices in the graphs of landscapes and state spaces "individuals" or "potential solutions" (or something else) is of little importance.

5.3.3. Fitness Functions and Heuristic Functions

The correspondence between landscapes and state spaces at the graph level was shown above. We now consider the labels on the vertices in the respective graphs. In the landscape model, graph vertices are labeled with elements of a partial order, $\mathcal{F}$, and the label on a vertex is a measure of the worth of that vertex as a solution to the search problem. If the vertex corresponds to more than a single element of $\mathcal{R}$, then the label attached to the vertex will typically be some overall measure of the worth of the points of $\mathcal{R}$ to which the vertex corresponds. In heuristic search, labels drawn from a partial order are also attached to states. The label is again a reflection of the worth of the set of solutions that is represented by that state. An approach similar to this is used to determine whether one set dominates another in search via dynamic programming. In other situations, the label attached to a state is interpreted as a measure of the distance to a goal state elsewhere in the graph. The importance of the relationship between fitness functions in GAs and heuristic functions that provide an estimate of the distance to a goal is explored below.

5.3.4. Navigation Strategies and Control Strategies

Search algorithms in AI contain a direct counterpart of what I have called the navigation strategy of an evolutionary algorithm. The method by which AI search algorithms explore their state space graphs has been termed the *control strategy* (e.g., [31, 44]). The quotation from Rich [44] at the beginning of Chapter 2 illustrates the extent to which AI search algorithms are viewed as being tied to the idea of exploration of a graph structure. The division of evolutionary algorithms into structure and navigation strategy is entirely in keeping with the view of search that AI has held for many years. The remainder of this section outlines some of the most common AI control strategies.

- **Generate-and-Test** was the name given to early control strategies. The name is not a particularly useful one as descriptions of generate-and-test methods tend to be so general that they encompass a very wide range of possible control strategies. The following description (adapted from [44])

  1. Generate a possible solution.
  2. Test to see if this is actually a solution.
  3. If a solution has been found, quit. Otherwise, return to step 1.

  does not specify at all how step 1 should be performed. The operation of the algorithm can be imagined to take place on a landscape graph $K_{1,n}$ with edges provided by the operator that implements step 1, though there is little to be gained from such visualization.

- In **Means-Ends Analysis** a problem that cannot be directly solved is broken into subproblems. The control strategy focuses on reducing the distance between the current state and the goal state (these can be thought of as nodes in a graph). Reduction into subproblems occurs when it is noticed that although the system does not know how to get directly from A to C, it could complete this if it knew how to get from A to B and subsequently from B to C. Identifying a suitable node B may not be simple. The production system uses a collection of rules consisting of preconditions and actions. The preconditions match the differences that need to be removed to bring one closer to a solution and the actions effect the removal. The best known use of means-ends analysis was in the *General Problem Solver* [136].

- **Production Systems** consist of three elements (from Nilsson [137, page 17]):

1. A *global database*,
2. A set of *production rules*, and
3. A *control system.*

The rules consist of preconditions and actions. The preconditions may match some aspect of the global database, in which case the rule may be applied to the database, thereby changing it. The control system determines which rules it is appropriate to invoke, resolves conflicts (when the preconditions of many rules are satisfied) and brings the search to a halt when the global database satisfies some stopping criterion. This description may seem to have little to do with evolutionary algorithms. Actually, it has *everything* to do with these algorithms. If one substitutes $\mathcal{C}$ (the collection of elements of the representation space in the landscape model), operators and navigation strategy for global database, production rules and control system in the above, the description fits the landscape model almost perfectly.

- **Bi-Directional Search.** When searching a graph that contains a known start and goals state (or states), a common technique is to try the search in both directions. More powerful, and more complicated, is to conduct the forward and backward searches simultaneously. This is known as *bi-directional* search [138]. In terms of the number of nodes visited, the potential gains from this method are enormous. If the number of nodes visited increases exponentially with search depth (as is often the case), and the two searches meet at a ideal distance close to halfway between their origins, the search requires that only $O(b^{d/2})$ nodes be visited instead of the $O(b^d)$ that would be required of a single uni-directional search. Naturally, such a control strategy relies on knowledge of the initial and goal states and that the problem is to find a path between these. Neither of these is typically true of the problems that are addressed by evolutionary algorithms.

- **Other Graph Searching Control Strategies.** There are many other control strategies that are in wide use. One of the best known is the A^* algorithm [139]. These algorithms are generally designed explicitly to search graphs. In many cases they interpret the value found at a vertex as an estimate of the distance to some goal vertex. Pearl provides extensive coverage of many such control strategies [31].

5.4. Differences Between Evolutionary Algorithms and State Space Search

The previous section emphasized many commonalities between heuristic state space search and evolutionary algorithms. The main difference between the two was that some of the control strategies were not applicable to evolutionary algorithms as the latter typically have no sense of start or goal nodes. Even if they encounter a satisfactory vertex, the problem statement may not provide enough information that this can be recognized. There are a number of other differences between these search algorithms.

Search algorithms in AI and OR are often not concerned with locating individuals. In fact, the optimal individual's identity and location are sometimes part of the problem statement (consider Rubik's Cube); the problem is to find a path through a graph that connects the current vertex to a goal vertex. Often, the problem contains some predefined connectivity. Also common is the search for a tree, perhaps a decision tree that represents a strategy for solving a problem. Most of the problems addressed by AI seem to fall into one of a few categories: search for an individual, search for a path and search for a tree. Pearl [31, page 26] calls these three constraint-satisfaction problems, path-finding problems and strategy-finding problems, and argues that a state space representation (an OR-graph) is generally a better representation of the first two and that a problem-reduction representation (AND/OR-graph) is generally better for the latter.

AI and OR search algorithms do not restrict themselves to trafficking exclusively in whole individuals, whereas evolutionary algorithms typically do. In many cases, partial individuals are created, compared and extended by these algorithms. As mentioned above, a partial solution can also be regarded as a set of solutions that all include the partial solution. This difference in viewpoint is historical; AI regards partial solutions as being generated and enhanced to form whole solutions, while OR regards them as representing sets of possible solutions to be split and pruned [31]. Branch-and-bound and dynamic programming algorithms manipulate these partial solutions explicitly, using comparisons based on lower bounds and dominance relations. A general treatment of search algorithms that operate on partial objects (there called *conjuncts*) can be found in [140], which uses TSP and the matrix chain association problem as detailed examples.

Compared to the similarities between heuristic state space search and evolutionary algorithms, these differences are rather minor. The commonalities are so strong that differences provide an indication of a possible area of fruitful interaction between the fields rather than an indication that the correspondence is weak. For example, AI and OR algorithms commonly operate on partial individuals but this is not the case in evolutionary algorithms. Rather than challenging the correspondence between the fields, one may be tempted to devise evolutionary algorithms that explicitly manipulate partial individuals. As another example, the values attached to vertices by an heuristic function in AI are often interpreted as a distance to a goal, but this is never done in evolutionary algorithms. The result of viewing the fitness function of an evolutionary algorithm as an heuristic (distance providing) function is the subject of the following sections.

5.5. Fitness and Heuristic Functions Revisited

We now return to the correspondence between fitness functions and heuristic functions. These functions are used to label the vertices in a graph. If we ask how this could be done to make search as simple as possible, one answer is the following. Suppose the set of vertices in the graph that represent an object that would satisfy the purposes of the search is S. Then label each vertex in the graph with its distance from the nearest element of S. This produces a landscape on which we would like to find vertices with low "fitness." Negating each fitness value converts the problem into one of maximization. Such a landscape is simple to search because from every vertex there is a path of increasing fitness that leads to an element of S. Actually, every uphill direction leads to an element of S. It is not hard to see that if we could find such a fitness function in polynomial time that we could solve the well-known NP-complete SATISFIABILITY problem in polynomial time [141]. If each of n binary variables is represented by a bit in a bit string of length n, then steepest ascent hillclimbing will examine n different assignments to the variables on each step of its search, and the search will take no more than n such steps, giving a worst-case of $O(n^2)$ different variable settings that need testing before a satisfying solution is found.

It is clearly unreasonable to hope that we might find a fitness function such as this in polynomial time (unless P=NP). However, it might be reasonable to expect that the more a fitness function resembles this ideal, the simpler the landscape will be to search. If this is true, we should be seeking fitness functions that do a good job of providing an estimate of the distance to a solution. A fitness function of this nature is not really a fitness function if we consider a fitness function to be something that provides a measure of the worth of an individual without reference to other individuals. The ideal fitness function just described does not provide this kind of information. Instead, it provides a measure of how far away a good individual is.

Such a fitness function is exactly what is sought in an heuristic function for many AI search algorithms. In these algorithms, the value attached to a vertex by the evaluation function is often interpreted as a distance. For example, in A* [139], search from a state n proceeds according to a function $f(n) = g(n) + h(n)$ where $g(n)$ is a function estimating the minimum distance from the starting state to state n and $h(n)$ is an heuristic function estimating the minimum distance from n to the goal state. There are many results that show that the better an estimate $h(n)$ is to the function $h^*(n)$, which gives the exact distance to the goal, the better an heuristic search algorithm will perform. AI is typically concerned with *admissible* heuristic functions, in which $\forall n, h(n) \leq h^*(n)$. However, the original descriptions of searching labeled graphs suggested only that $h(n)$ be correlated with $h^*(n)$ [142].

The type of fitness function we would most like in evolutionary algorithms is exactly what is desirable as an heuristic function in AI search algorithms. If we assume that the closer our fitness functions approximate the AI ideal the easier search will be, and can quantify how well they do this, we have a measure of search difficulty. The usefulness of the measure will provide an indication of how realistic the original assumption was. The remainder of this chapter introduces a measure of this correlation and investigates its usefulness as a predictor of GA performance.

5.6. GA Difficulty

The search for factors affecting the ability of the GA to solve optimization problems has been a major focus within the theoretical GA community. Horn and Goldberg [143] recently stated “If we are ever to understand how hard a problem GAs can solve, how quickly, and with what reliability, we must get our hands around what ‘hard’ is.” The most visible attempt at pinning down what it is that makes search more or less difficult for GAs is work on the theory of *deceptive* functions, which has been developed by Goldberg [104] and others, based upon early work by Bethke [144]. However, researchers seem quite divided over the relevance of deception, and we have seen reactions ranging from proposals that “the only challenging problems are deceptive” [145, 146] to informal claims that deception is irrelevant to real-world problems, and Grefenstette’s demonstration that the presence of deception is not necessary or sufficient to ensure that a problem is difficult for a GA [147]. In addition, the approach has not been generalized to GAs that do not operate on binary strings; it requires complete knowledge of the fitness function; quantifying deception can be difficult computationally; the theory is rooted in the schema theorem, which has also been the subject of much recent debate; and finally, non-deceptive factors such as spurious correlations (or hitch-hiking) [96, 148] have been shown to adversely affect the GA. Kargupta and Goldberg have recently considered how signal and noise combine to affect search [149, 150]. They focus on how the dynamics of schema processing during the run of a GA alters measures of signal and noise. This method is promising, and provides plausible explanations for GA performance on a number of problems, some of which are considered here.

Another attempt to capture what it is that makes for GA difficulty is centered around the notion of "rugged fitness landscapes." At an informal level, it is commonly held that the more rugged a fitness landscape is, the more difficult it is to search. While this vague statement undoubtedly carries some truth, "ruggedness" is not easily quantified, even when one has defined what a landscape is. Unfortunately, the informal claim also breaks down. For example, Ackley [26] and Horn and Goldberg [143] have constructed landscapes with a provably maximal number of local optima, but the problems are readily solved by a GA. At the other extreme, a relatively smooth landscape may be maximally difficult to search, as in "needle in a haystack" problems. Thus, even before we can define what ruggedness might mean, it is clear that our intuitive notion of ruggedness will not always be reliable as an indicator of difficulty and we can expect that it will be extremely difficult to determine when the measure is reliable.

The most successful measure of ruggedness developed to date has been the calculation of "correlation length" by Weinberger [65] which was the basis for the work of Manderick et al. [60]. Correlation length is based on the rate of decrease in correlation between parent and offspring fitness, and clearly suffers from the above problem with relatively flat landscapes—correlation length is large (indicating an easy search problem) but the problem may be very difficult. Additionally, parent/offspring fitness correlation can be very good even when the gradient of the landscape is leading away from the global maximum, as in deceptive problems. Associated with ruggedness is the notion of "epistatic interactions," which were the basis of a proposed viewpoint on GA difficulty proposed by Davidor [151]. Although it is clear that some highly epistatic landscapes are difficult to search [117, 118], it is not clear how much epistasis is needed to make a problem difficult and Davidor's measure is not normalized, making comparisons between problems difficult. Also, the method does not provide confidence measures; is computationally "not economical," and will have the same problems on landscapes with little or no epistasis (because they are relatively flat) as described above.

As final testimony to the claim that we have not yet developed a reliable indicator of GA hardness, there have been several surprises when problems did not prove as easy for a GA as had been expected. Tanese [152] constructed a class of Walsh polynomials of fixed order and found that a GA encountered difficulty even on theoretically easy low-order polynomials. In an attempt to study GA performance in a simple environment, Mitchell et al. constructed the "royal road" functions [36]. They compared the GA's performance on two royal road functions, one of which contained intermediate-size building blocks that were designed to lead the GA by the hand to the global optimum. Surprisingly, the GA performed better on the simpler function in which these intermediate building blocks did not exist.

All these notions of what makes a problem hard for a GA all have something to recommend them, but all seem to be only a piece of the whole story. It is clear that we are still some way from a good intuition about what will make a problem hard for a GA. I propose that it is the relationship between fitness and distance to the goal that is important for GA search. This relationship is apparent in scatter plots of fitness versus distance and is often well summarized by computing the correlation coefficient between fitness and distance.

5.7. Fitness Distance Correlation

The easiest way to measure the extent to which the fitness function values are correlated with distance to a global optimum is to examine a problem with known optima, take a sample of individuals and compute the correlation coefficient, r, given the set of (fitness, distance) pairs. As we are maximizing, we should hope that fitness increases as distance to a global maxima decreases. With an ideal fitness function, r will therefore be -1. When minimizing, the ideal fitness function will have $r = 1$. Given a set $F = \{f_1, f_2, \cdots, f_n\}$ of n individual fitnesses and a corresponding set $D = \{d_1, d_2, \cdots, d_n\}$ of the n distances to the nearest global maximum, we compute the correlation coefficient r, in the usual way, as

$$r = \frac{C_{FD}}{\sigma_F \sigma_D}$$

where

$$C_{FD} = \frac{1}{n}\sum_{i=1}^{n}(f_i - \overline{f})(d_i - \overline{d})$$

is the covariance of F and D, and σ_F, σ_D, $\overline{f}$ and $\overline{d}$ are the standard deviations and means of F and D. This fitness distance correlation will be abbreviated FDC.

In the results that follow, Hamming distances are always used and the distance associated with an individual is the distance from it to the closest point which satisfies the object of the search (usually a global maxima). It is likely that a better statistic would be obtained if distances were computed using the operator that defined the edges of the landscape graph, though these will be more difficult to compute. Hamming distance is a simple first approximation to distance under the actual operators of a GA. That Hamming distance works well as a predictor of GA performance is perhaps a result of its close relationship to distance under mutation. These issues are discussed in §5.7.2(158).

I will use r (FDC) as a measure of problem difficulty. Correlation works best as a summary statistic of the relationship between two random variables if the variables follow a bivariate normal distribution. There is no guarantee that this will be the case if we have a random sample of fitnesses, and there are therefore situations in which r will be a poor summary statistic of the relationship between fitness and distance. I am *not* claiming that correlation is necessarily a good way to summarize the relationship between fitness and distance. I do claim that this relationship is what is important. In practice, examining the scatter plot of fitness versus distance is very informative in the cases where there is a structure in this relationship that cannot be detected by correlation. It is important to realize that correlation is only one of the possible ways that the relationship between fitness and distance can be examined. It appears quite useful, although we will see examples of problems for which it is too simplistic. In all of these cases, the scatter plots are useful for revealing the shortcomings of correlation.

5.7.1. Summary of Results

The reliability of the FDC measure is explored by first confirming existing knowledge about a wide range of problems. The results obtained have been uniformly encouraging. I chose to test FDC in three ways: (1) to confirm existing knowledge about the behavior of a GA on a number of reasonably well-studied problems, (2) to test whether FDC would have predicted results that at one time seemed surprising, and (3) to investigate whether FDC could detect differences in coding and representation. These are discussed in three subsections below. In the results that follow, FDC is computed via examining all points in the space when it contains 2^{12} or fewer points, and via a sample of 4,000 randomly chosen points otherwise. Given a binary string x, the number of ones in x will be denoted by $u(x)$.

Figure 53 shows r values for some instances of all the problems studied. The vertical axis corresponds to the value of r obtained. Problems that are treated together are grouped together on the horizontal axis. An explanation of the abbreviations used

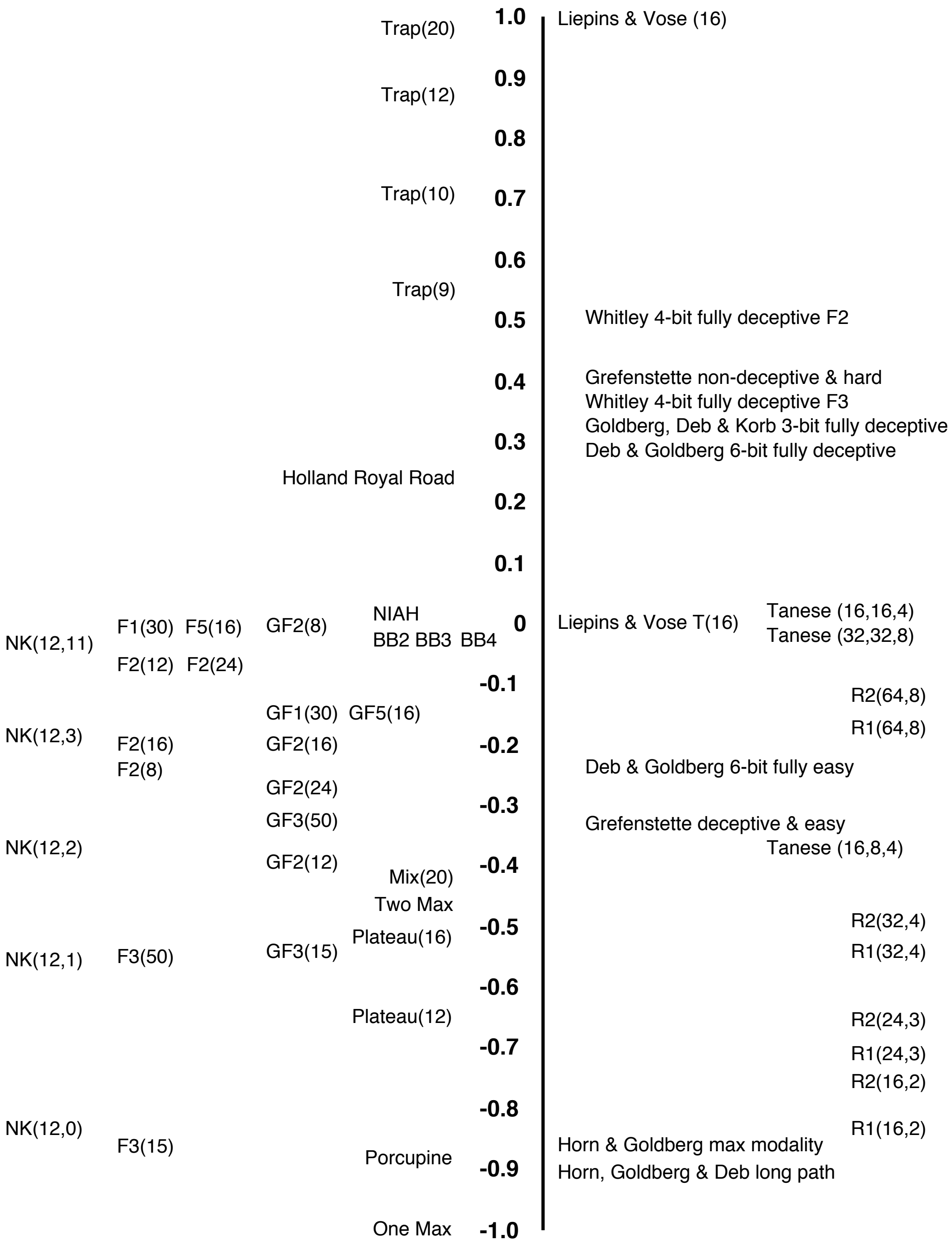


Figure 53. Summary of results. Horizontal position is merely for grouping, vertical position indicates the value of r. Abbreviation explanations and problem sources are given in Table 32.

in this figure, together with a short description of the problems and their sources can be found in Table 32. Figures 54 and 104 show examples of scatter plots of fitness and distance from which r is computed. The plots represent all the points in the space unless a number of samples is mentioned. In these scatter plots, a small amount of noise has been added to distances (and in some cases fitnesses) so that identical fitness/distance pairs can be easily identified. This was suggested by Lane [153] and in many cases makes it far easier to see the relationship between fitness and distances. This noise was not used in the calculation of r, it is for display purposes only.

5.7.1.1. Confirmation of Known Results

This section investigates the predictions made by FDC on a number of problems that have been relatively well-studied. These include various deceptive problems, and other simply defined problems.

Easy Problems

Ackley's "one max" problem [26] was described in §3.8.1(58). According to the FDC measure, this problem is as simple as a problem could be. It exhibits perfect negative correlation ($r = -1$), as is shown in Figure 54. The one max fitness function is essentially the ideal fitness function described above. The distance to the single global optimum is perfectly correlated with fitness. Ackley's "two max" problem is also correctly classified as easy by FDC ($r = -0{\cdot}41$). This function has two peaks, both with large basins. For binary strings of length n, the function is defined as

$$f(x) = |18u(x) - 8n|.$$

For $K < 3$, the NK landscape problems (described in §4.7.1(97)) produce high negative correlation ($-0{\cdot}83$, $-0{\cdot}55$ and $-0{\cdot}35$), though r moves rapidly towards zero as K increases, which qualitatively matches the increases in search difficulty found by Kauffman [117, 118] and others. As NK landscapes are constructed from a table of $N2^{K+1}$ random numbers, the r value for each K value is the mean of ten different landscapes. Figures 56 and 58 show three NK landscapes for $N = 12$ and $K = 1$, 3 and 11. When $K = 11$ the landscape is completely random, and this is reflected by an r value that is very close to zero.

One, two and three instances of Deb and Goldberg's [105] 6-bit fully easy problem (described in §3.8.2(59)) are shown in Figures 59 and 61. Interestingly, each of these problems has $r = -0{\cdot}2325$, which is an indication that in some sense the problem difficulty is not affected by changing the number of copies of the same subfunction.

Table 32. The problems of Figure 53. Where a problem has two sources, the first denotes the original statement of the problem and the second contains the description that was implemented.

Abbreviation	Problem Description	Source
BBk	Busy Beaver problem with k states.	[106, 110]
Deb & Goldberg	6-bit fully deceptive and easy functions.	[105]
F$k(j)$	De Jong's function k with j bits.	[30, 5]
GF$k(j)$	As above, though Gray coded.	[30, 5]
Goldberg, Korb & Deb	3-bit fully deceptive.	[154, 145]
Grefenstette easy	The deceptive but easy function.	[147]
Grefenstette hard	The non-deceptive but hard function.	[147]
Holland royal road	Holland's 240-bit royal road function.	[112, 155]
Horn, Goldberg & Deb	The long path problem with 40 bits.	[86]
Horn & Goldberg	A 33-bit maximally rugged function.	[143]
Liepins & Vose (k)	Deceptive problem with k bits.	[47, 156]
Mix(n)	Ackley's mix function on n bits.	[26]
NIAH	Needle in a haystack.	p. 148
NK(n, k)	Kauffman's NK landscape. N = n, K = k.	[117]
One Max	Ackley's single β-peaked function.	[26]
Plateau(n)	Ackley's plateau function on n bits.	[26]
Porcupine(n)	Ackley's porcupine function on n bits.	[26]
R(n, b)	Mitchell et al. n-bit royal road, b-bit blocks.	[36, 96]
Tanese (l, n, o)	l-bit Tanese function of n terms, order o.	[152, 157]
Trap(n)	Ackley's trap function on n bits.	[26]
Two Max(n)	Ackley's two β-peaked function on n bits.	[26]
Whitley Fk	4-bit fully deceptive function k.	[145]

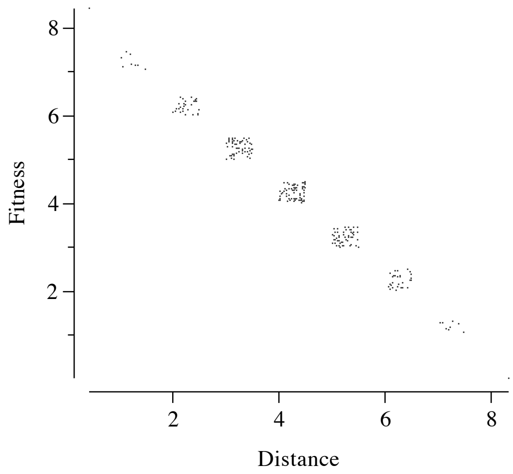


Figure 54. Ackley's one max problem on 8 bits ($r = -1$). Ackley's fitnesses were actually ten times the number of ones, which has no affect on r.

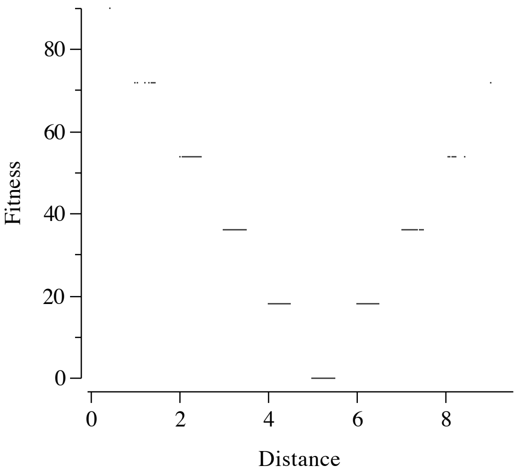


Figure 55. Ackley's two max problem on 9 bits ($r = -0{\cdot}41$).

Although an algorithm may require more resources to solve a problem with more subfunctions, the problem difficulty, in the eyes of FDC, does not increase. This is intuitively appealing. It is very hard to dissociate problem difficulty from algorithm resources, yet the FDC measure can be interpreted as "recognizing" that solving the same problem twice (perhaps simultaneously) is no harder than solving it once. Of course it will require more resources, but it can be argued that it is not more difficult. It is possible to prove that FDC remains the same when any number of copies of a function are concatenated in this manner. For this reason, we will see this behavior in all the problems below that involve multiple identical subproblems. The proof of this invariance is given in Appendix D.

Figure 62 shows Grefenstette's deceptive but easy problem [147] ($r = -0{\cdot}33$). In this problem, two variables x_1 and x_2 are encoded using 10 bits each. The problem is to maximize

$$f(x_1, x_2) = \begin{cases} x_1^2 + 10x_2^2 & \text{if } x_2 < 0{\cdot}995. \\ 2(1 - x_1)^2 + 10x_2^2 & \text{if } x_2 \geq 0{\cdot}995. \end{cases}$$

While this problem is highly deceptive (at least under some definitions of deception), it is simple for a GA to optimize. FDC correctly classifies the problem as simple.

Ackley's maximally rugged "porcupine" function is shown in Figure 64. In this

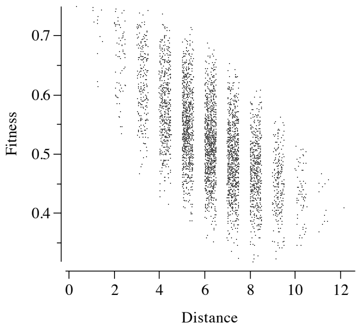


Figure 56. An NK landscape with $N = 12$ and $K = 1$ ($r = -0\cdot64$).

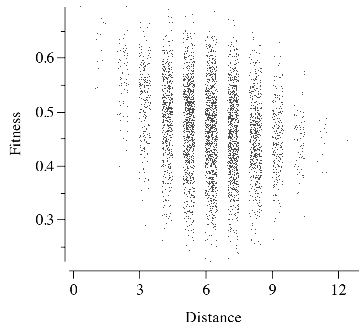


Figure 57. An NK landscape with $N = 12$ and $K = 3$ ($r = -0\cdot25$).

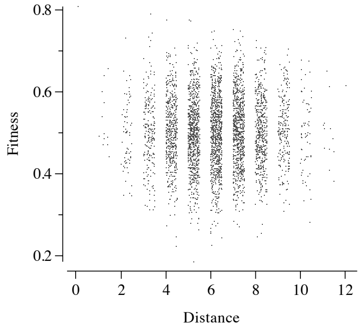


Figure 58. An NK landscape with $N = 12$ and $K = 11$ ($r = -0\cdot01$).

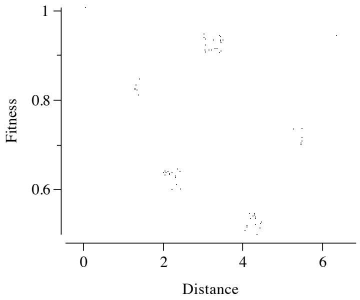


Figure 59. Deb & Goldberg's fully easy 6-bit problem ($r = -0\cdot23$).

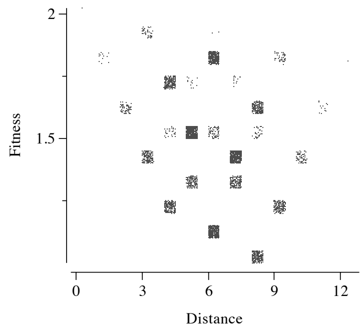


Figure 60. Two copies of Deb & Goldberg's fully easy 6-bit problem ($r = -0\cdot23$).

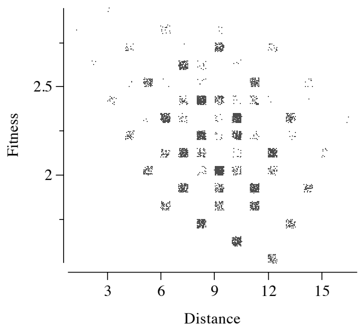


Figure 61. Three of Deb & Goldberg's fully easy 6-bit problems ($r = -0\cdot23$, 4000 sampled points).

function, every binary string with even parity is a β-local-maximum. For a binary string of even length n, the function is defined as

$$f(x) = \begin{cases} 10u(x) - 15 & \text{if } u(x) \text{ is odd.} \\ 10u(x) & \text{otherwise.} \end{cases}$$

Despite the ruggedness, the function is not difficult to optimize (unless your algorithm never goes downhill and only employs an operator that changes a single bit at a time). FDC is very strongly negative ($r = -0\cdot88$).

Horn and Goldberg's maximally rugged function, shown in Figure 65 is very similar and also exhibits strong negative correlation ($r = -0\cdot83$). Given a binary

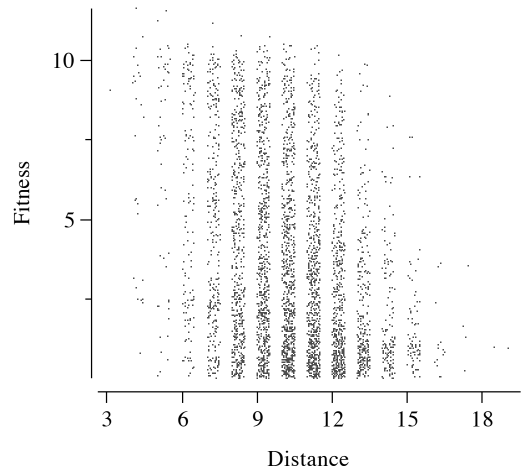


Figure 62. Grefenstette's deceptive but easy 20-bit problem ($r = -0{\cdot}32$, 4000 sampled points).

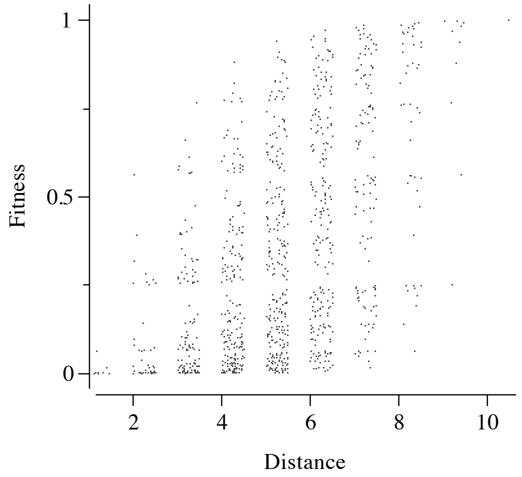


Figure 63. Grefenstette's non-deceptive but hard 10-bit problem. The single point with fitness 2048 is omitted from the plot. When included, $r = -0{\cdot}09$, when excluded, $r = 0{\cdot}53$).

string, x, of odd length, the function is defined to be

$$f(x) = u(x) + 2 \times \mathit{parity}(x)$$

where $\mathit{parity}(x)$ is one if $u(x)$ is odd and zero otherwise. It is easy to see that in this function, every binary string with odd parity is a β-local-optimum. The string with all ones is the global optimum. This function is known to be easily optimized by a GA [143].

These two maximally rugged functions provide examples in which the idea that a lot of ruggedness will make search difficult is incorrect. On both of them, FDC reports correctly that the problems should not be difficult.

Finally, Ackley's "mix" function, a combination of five functions studied in [26] (all of which are considered here) is also classified as easy by FDC. This is not surprising as four of his five functions receive strong negative r values. The function is a combination of "one max," "two max," "trap," "porcupine," and "plateaus." The trap and plateaus functions are introduced below. Figure 66 shows the scatter plot from a sample of points on the 20-bit mix function.

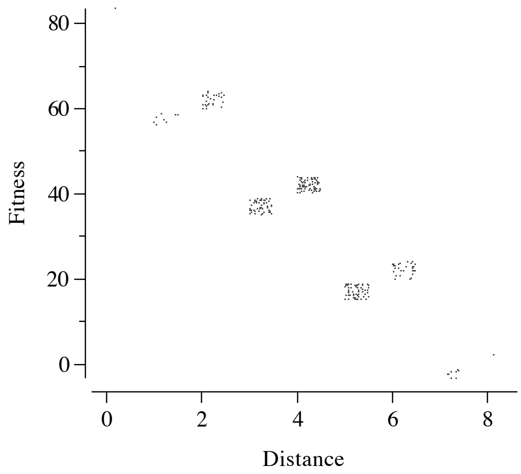


Figure 64. Ackley's porcupine problem on 8 bits ($r = -0{\cdot}88$).

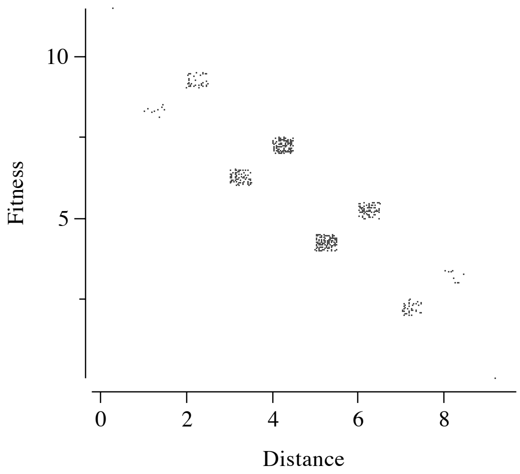


Figure 65. Horn & Goldberg's maximum modality problem on 9 bits ($r = -0{\cdot}83$).

Deceptive Problems

An early fully deceptive problem is that given by Goldberg, Korb and Deb [154]. It is defined over three bits as follows:

$$\begin{array}{ll} f(000) = 28 & f(100) = 14 \\ f(001) = 26 & f(101) = 0 \\ f(010) = 22 & f(110) = 0 \\ f(011) = 0 & f(111) = 30 \end{array}$$

Figures 67 and 69 show two to four concatenated copies of this function for which $r = 0{\cdot}32$. Deb and Goldberg's 6-bit fully deceptive problem [105] (described in §3.8.3(60)) has $r = 0{\cdot}30$. Figures 70 and 72 show one to three copies of this function. As with the fully easy subproblems described above, the concatenation of several deceptive subproblems does not affect r.

Ackley's "trap" function is defined on binary strings of length n as follows

$$f(x) = \begin{cases} (8n/z)(z - u(x)) & \text{if } u(x) \leq z. \\ (10n/(n-z))(u(x) - z) & \text{otherwise.} \end{cases}$$

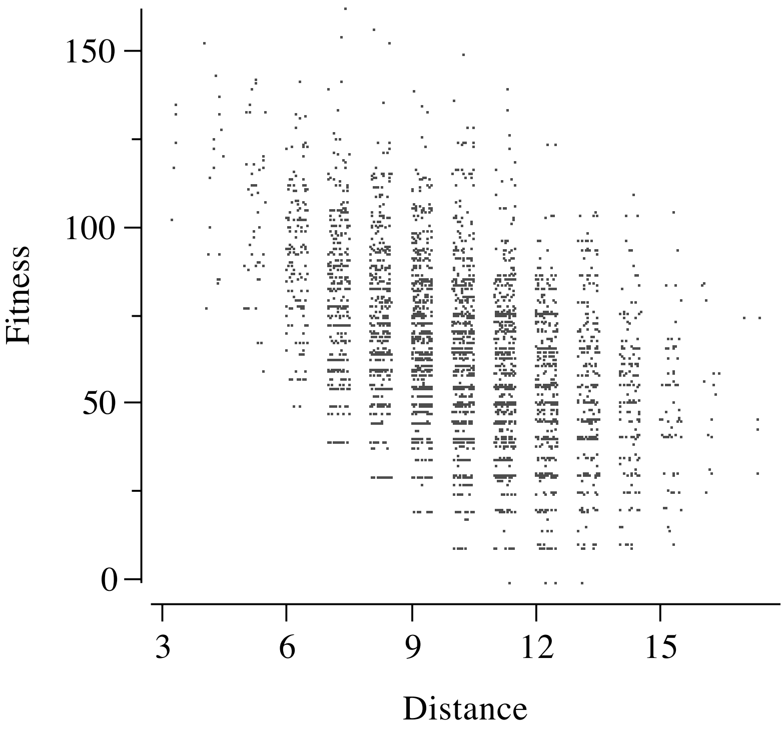


Figure 66. Ackley's mix problem on 20 bits ($r = -0{\cdot}44$, 4000 sampled points).

where $z = \lfloor 3n/4 \rfloor$. This function has two β-peaks the higher of which has a small basin of attraction. Deb and Goldberg showed that the function is not fully deceptive for $n < 8$ [158]. However, the problem becomes increasingly difficult (from the point of view of β) as n increases, as the basin of attraction (under β) of the global maximum (the string with n ones) only includes those points with $u(x) > z$, a vanishingly small fraction of the entire space [41, page 22]. FDC becomes increasingly strongly negative as n is increased. For $n = 9$, 10, 12 and 20, the r values obtained are 0·56, 0·71, 0·88 and 0·98 respectively. The last of these is from a sample of 4,000 points and the scatter plot is shown in Figure 73.

In [145], Whitley discusses three fully deceptive functions. The first of these is the 3-bit problem due to Goldberg, Korb and Deb [154] just discussed. The second and third functions (which will be referred to as Whitley's F2 and F3) both had four

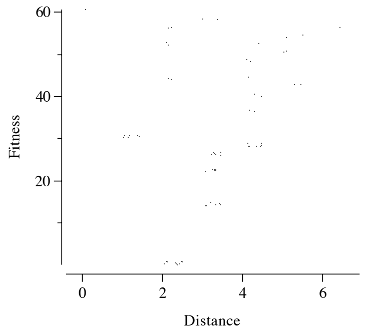


Figure 67. Two copies of Goldberg, Deb & Korb's fully deceptive 3-bit problem ($r = 0{\cdot}32$).

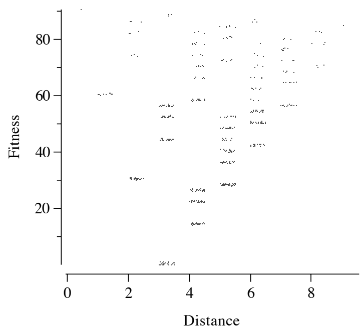


Figure 68. Three copies of Goldberg, Deb & Korb's fully deceptive 3-bit problem ($r = 0{\cdot}32$).

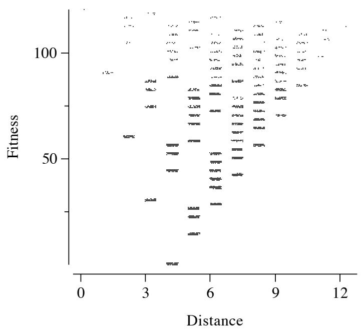


Figure 69. Four copies of Goldberg, Deb & Korb's fully deceptive 3-bit problem ($r = 0{\cdot}32$).

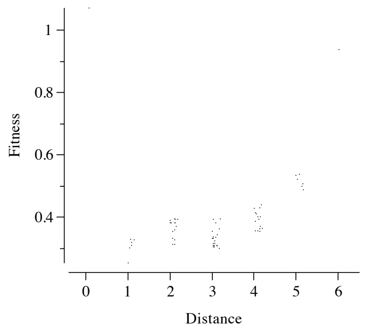


Figure 70. Deb & Goldberg's fully deceptive 6-bit problem ($r = 0{\cdot}30$).

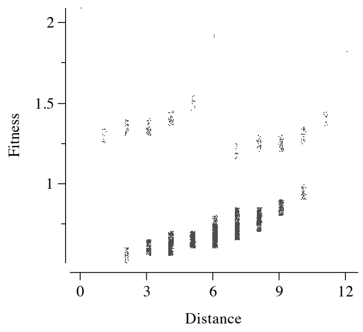


Figure 71. Two copies of Deb & Goldberg's fully deceptive 6-bit problem ($r = 0{\cdot}30$).

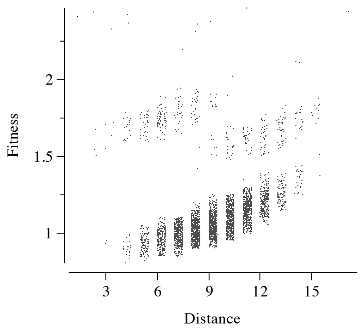


Figure 72. Three copies of Deb & Goldberg's fully deceptive 6-bit problem ($r = 0{\cdot}30$, 4000 sampled points). Notice the additive effect.

bits. The function values for the 16 four bit strings for F2 are as follows:

$$\begin{array}{llll}
f(0000) = 28 & f(0100) = 22 & f(1000) = 20 & f(1100) = 8 \\
f(0001) = 26 & f(0101) = 16 & f(1001) = 12 & f(1101) = 4 \\
f(0010) = 24 & f(0110) = 14 & f(1010) = 10 & f(1110) = 6 \\
f(0011) = 18 & f(0111) = 0 & f(1011) = 2 & f(1111) = 30
\end{array}$$

For one, two and three copies of Whitley's F2, $r = 0{\cdot}51$. Scatter plots of these are shown in Figures 74 and 76.

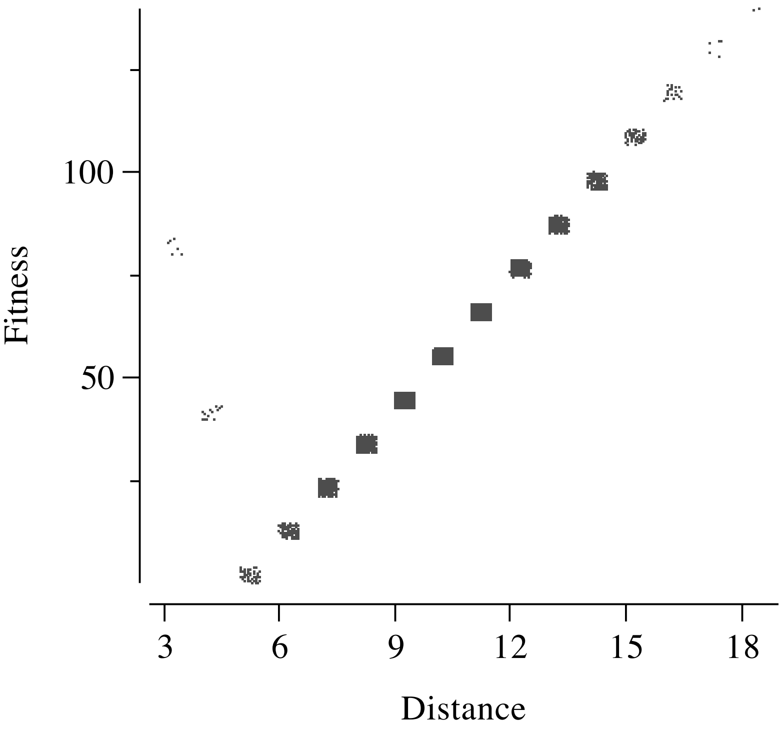


Figure 73. Ackley's trap function on 20 bits ($r = 0{\cdot}98$, 4000 sampled points).

The function values for Whitley's F3 are:

$$\begin{array}{llll} f(0000)=10 & f(0100)=27 & f(1000)=28 & f(1100)=5 \\ f(0001)=25 & f(0101)=5 & f(1001)=5 & f(1101)=0 \\ f(0010)=26 & f(0110)=5 & f(1010)=5 & f(1110)=0 \\ f(0011)=5 & f(0111)=0 & f(1011)=0 & f(1111)=30 \end{array}$$

for which $r = 0{\cdot}36$. Figures 77 and 79 show scatter plots of these functions.

Grefenstette's difficult but non-deceptive problem [147] gave $r = 0{\cdot}38$ and is shown in Figure 63. In this problem, a bit string of length L is used to encode the interval [0..1]. Then, given x (the real value obtained by decoding a binary string of

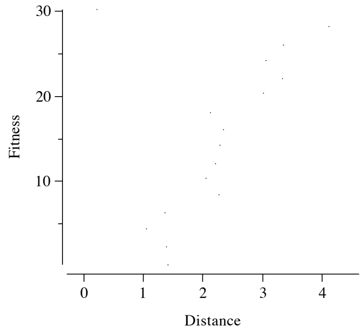


Figure 74. Whitley's F2. A fully deceptive 4-bit problem ($r = 0{\cdot}51$).

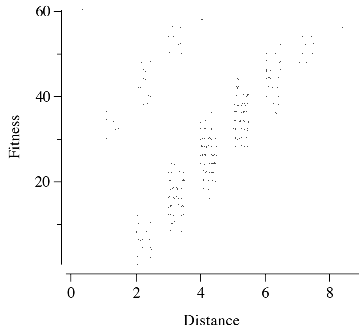


Figure 75. Two copies of Whitley's F2 ($r = 0{\cdot}51$).

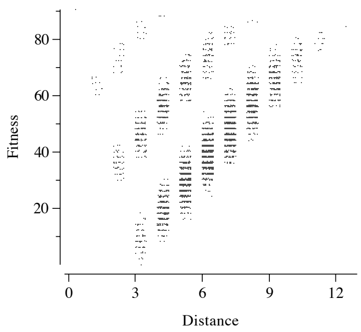


Figure 76. Three copies of Whitley's F2 ($r = 0{\cdot}51$).

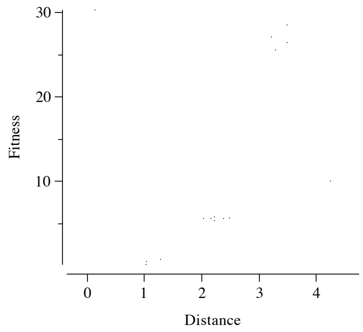


Figure 77. Whitley's F3. A fully deceptive 4-bit problem ($r = 0{\cdot}37$).

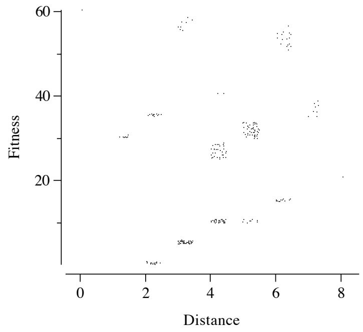


Figure 78. Two copies of Whitley's F3 ($r = 0{\cdot}37$).

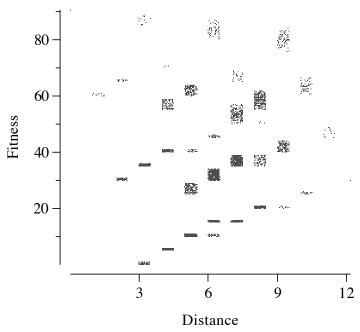


Figure 79. Three copies of Whitley's F3 ($r = 0{\cdot}37$).

length L), The function is defined as:

$$f(x) = \begin{cases} 2^{L+1} & \text{if } x = 0. \\ x^2 & \text{otherwise.} \end{cases}$$

FDC correctly classifies this problem as difficult.

Holland's royal road [112] (described in §3.8.6(62)) gives $r = 0{\cdot}26$ with parameter settings similar to Holland's defaults. Two members of Holland's class of functions are shown in Figures 80 and 81.

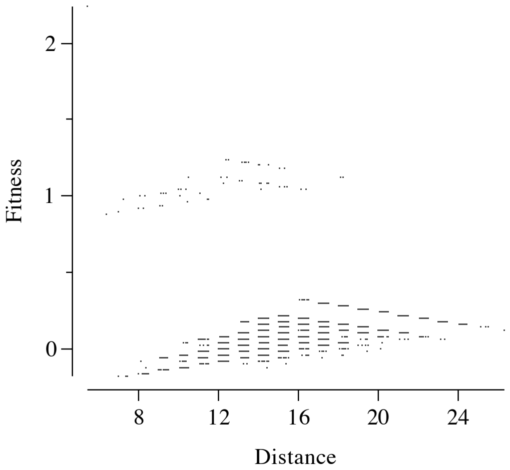


Figure 80. Holland's royal road on 32 bits ($b = 8$, $k = 2$ and $g = 0$), ($r = 0{\cdot}25$, 4000 sampled points).

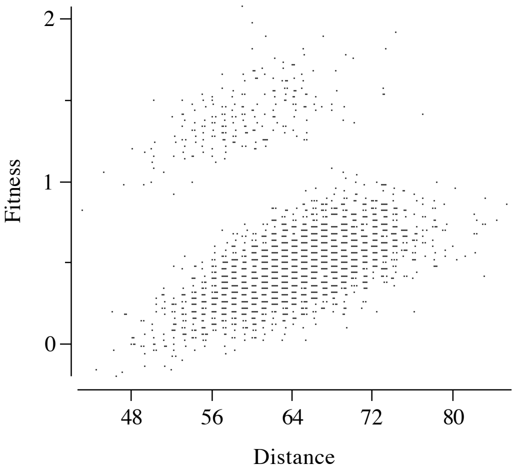


Figure 81. Holland's royal road on 128 bits ($b = 8$, $k = 4$ and $g = 0$), ($r = 0{\cdot}27$, 4000 sampled points).

Long Path Problems

Horn, Goldberg and Deb's long path problem [86] has $r = -0{\cdot}90$. This problem is difficult for the hillclimber it was constructed to be difficult for, but GAs apparently have little trouble with it. The idea of the function is to construct a path through the space of bit strings of a length n. The Hamming distance between successive points on the path is k and no point on the path is less than k bits away from any other point on the path. The path's length is exponential in n and the fitnesses assigned to points on the path are strictly increasing. The highest point on the path is the global optimum of the entire space. All points not on the path are placed on a slope that leads to some point on the path. As a result, a randomly started hillclimb that considers neighbors at distances not greater than k will at some point encounter the path and begin to follow it upwards. On average, the climb will have to traverse half the path and will thus involve an exponential number of steps. Thus, although the space has a single global optimum and no local optima, such a hillclimber has exponential average case performance. This runs counter to the intuition that a unimodal space is necessarily simple to search. This problem may be easy to an algorithm that uses an operator that modifies more than k bits in an individual. When $k = 1$, the GA does not have any great difficulty in finding the global optimum.

The r value above was obtained through sampling. When r is calculated for the entire space, its value falls considerably (towards 0). If we calculate r just for the

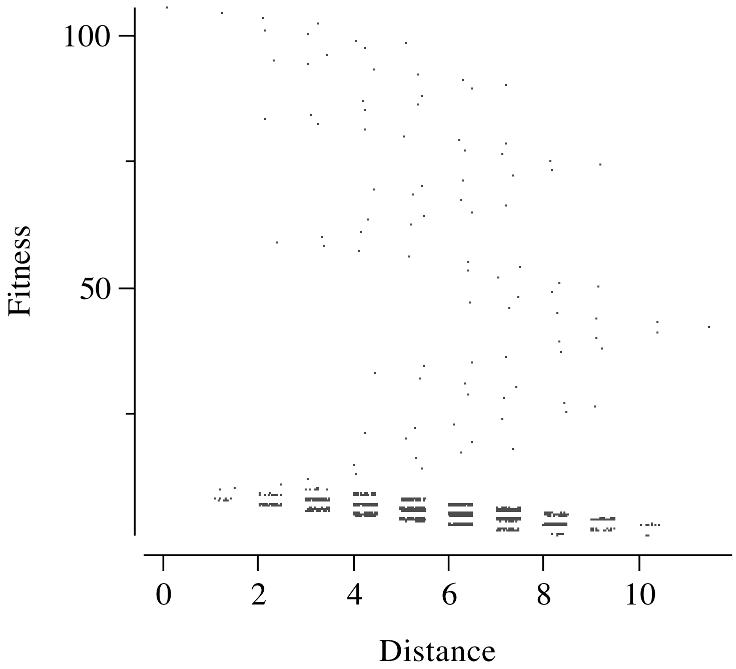


Figure 82. Horn, Goldberg & Deb's long path problem with 11 bits ($r = -0{\cdot}12$). Notice the path.

points on the path, it is strongly negative, as it is for the points not on the path (e.g., for strings with 12 bits, we get $r = -0{\cdot}39$ and $r = -0{\cdot}67$) but combining the samples gives a much lower correlation ($r = -0{\cdot}19$ for 12 bits). This is a first illustration of how correlation may sometimes prove too simplistic a summary statistic of the relationship between fitness and distance. Fortunately, the striking structure of the problem is immediately apparent from the scatter plot, as can be seen in Figure 82.

Zero Correlation

The needle in a haystack (NIAH) problem is defined on binary strings of length n as follows:

$$f(x) = \begin{cases} 1 & \text{if } u(x) = n. \\ 0 & \text{otherwise.} \end{cases}$$

The function is everywhere zero except for a single point. When we compute FDC exhaustively (or in a sample which includes the needle), r is very close to zero. This illustrates how FDC produces the correct indication when a function is very flat.

In such cases, measures such as correlation length will indicate that the problem is simple whereas it is actually maximally hard. If the sample used to compute FDC does not include the needle, the correlation is undefined as there is no variance in fitness. In this case it can be concluded that the problem is difficult (i.e., similar to $r = 0$) or some amount of noise can be added to fitness values, which will also result in $r \approx 0$. A similar NIAH problem is the following,

$$f(x) = \begin{cases} 100 & \text{if } u(x) = n. \\ \mathit{uniform}[0..1] & \text{otherwise.} \end{cases}$$

where $\mathit{uniform}[0..1]$ is a function that returns a real value chosen uniformly at random from the interval $[0..1]$. For the NIAH problem, r will be approximately zero whether the needle is sampled or not.

The 2-, 3-, and 4-state busy beaver problems (described in §3.8.5(60)) (Figures 83 and 85) and the NK(12,11) landscape (Figure 58 on page 140), all known to be difficult problems, also had r approximately 0.

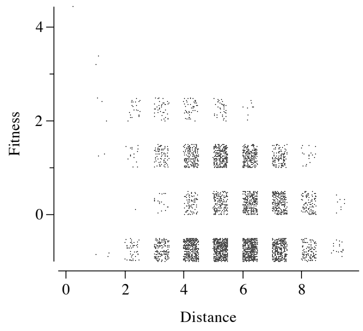


Figure 83. 2-state busy beaver problem ($r = -0{\cdot}06$, 4000 sampled points).

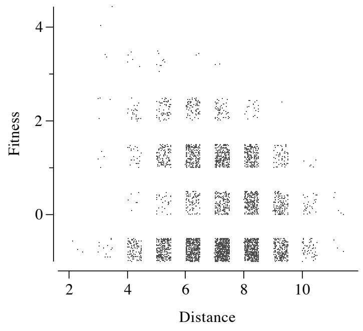


Figure 84. 3-state busy beaver problem ($r = -0{\cdot}08$, 4000 sampled points).

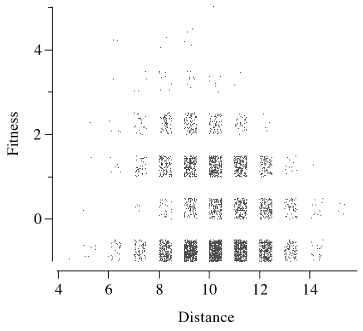


Figure 85. 4-state busy beaver problem ($r = -0{\cdot}11$, 4000 sampled points).

Some of De Jong's functions have very low r values. For example, F1(15) (Figure 86) has $r = -0{\cdot}01$. Though the correlation measures correctly predict that F1(15) will be harder for the GA than GF1(15), the low correlation for F1(15) is misleading. Looking at the scatter plot in Figure 86 it is clear that the function contains many highly fit points at all distances from the global optimum. From this, it is reasonable to expect that a GA will have no trouble locating a very high fitness point.

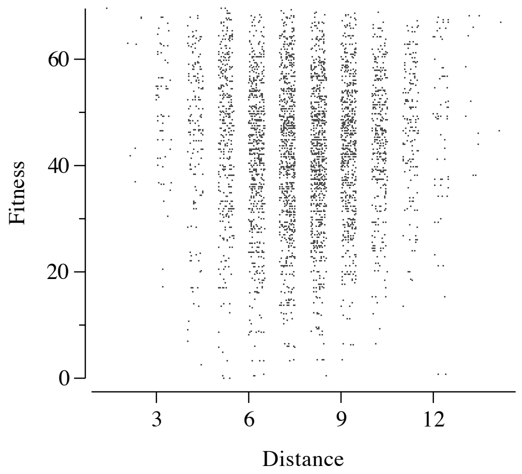


Figure 86. De Jong's F1 binary coded with 15 bits converted to a maximization problem ($r = -0{\cdot}01$, 4000 sampled points).

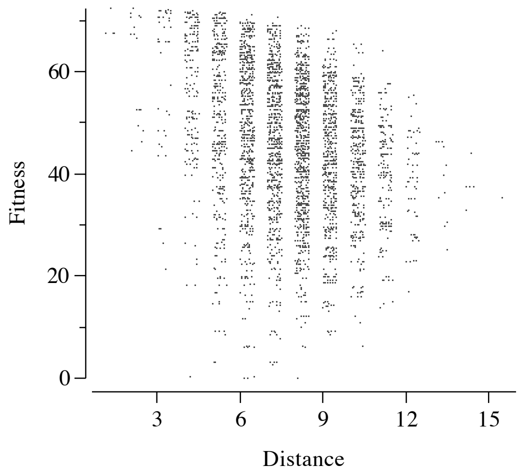


Figure 87. De Jong's F1 Gray coded with 15 bits ($r = -0{\cdot}30$, 4000 sampled points).

5.7.1.2. Confirmation of Unexpected Results

Here we examine the predictions of FDC on two problems sets whose results were surprising to GA researchers at the time they were obtained. These are the Tanese functions and the royal road functions. In both cases, FDC's predictions match the behavior of the GA.

Tanese Functions

Tanese found that on Walsh polynomials on 32 bits with 32 terms each of order 8 (which we will denote via T(32,32,8)) a standard GA found it very difficult to locate a global optimum [152]. FDC gives an r value very close to 0 for all the instances of T(32,32,8) considered, as it does for T(16,16,4) functions. When the number of terms is reduced, the problem becomes far easier, for instance, T(16,8,4) functions typically have an r value of approximately $-0{\cdot}37$. This contrast is illustrated by Figures 88 and 89. These results are consistent with the experiment of Forrest and Mitchell [157] who found that increasing string length made the problem far simpler in T(128,32,8) functions. It is not practical to use FDC on a T(128,32,8) function since it is possible to show that they have at least 2^{96} global optima and the FDC algorithm requires computing the distance to the nearest optimum.

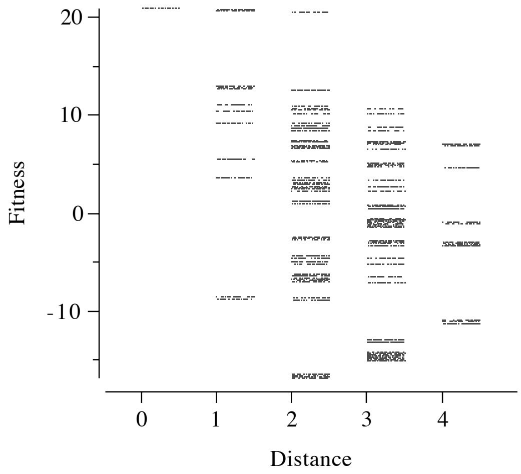


Figure 88. Tanese function on 16 bits with 8 terms of order 4 (r = −0·37, 4000 sampled points).

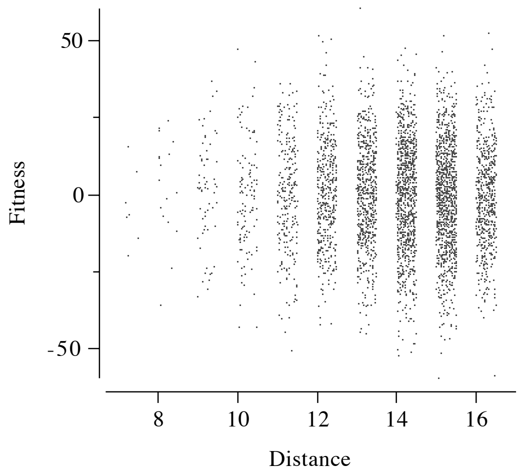


Figure 89. Tanese function on 32 bits with 32 terms of order 8 (r = 0·03, 4000 sampled points).

Royal Road Functions

Mitchell, Forrest and Holland examined the performance of a GA on two "royal road" problems, R1 and R2 [36]. Under R1 a 64-bit string was rewarded if it was an instance of 8 non-overlapping order 8, defining length 8, schema. R2 gives additional reward for instances of 4 order 16 and 2 order 32 schema that will ideally be discovered through recombination of lower order schema. It was thought that the additional building blocks of R2 would make the problem simpler for a GA. The opposite proved true. The GA performed slightly better on R1. FDC could have been used to predict this, or at least predict that R2 would not be simpler than R1. On a range on royal road functions, including the originals, R1 has had a slightly smaller r value than R2. Perhaps due to insufficient sampling, the difference does not appear significant on the original (64,8) problem. However it clearly is (at the 0·0001 confidence level with a Wilcoxon rank-sum test) on (32,4), (24,3) and (16,2) problems. Figures 90 and 91 illustrate this slight difference, showing R1(32,4) and R2(32,4).

It is interesting to compare the scatter plots from the royal road functions with those for Ackley's "plateaus" function shown in Figure 92. The plateau function divides a string into four equal sections and these are each rewarded only if they consist entirely of one bits. This is very similar to the R1 function described above.

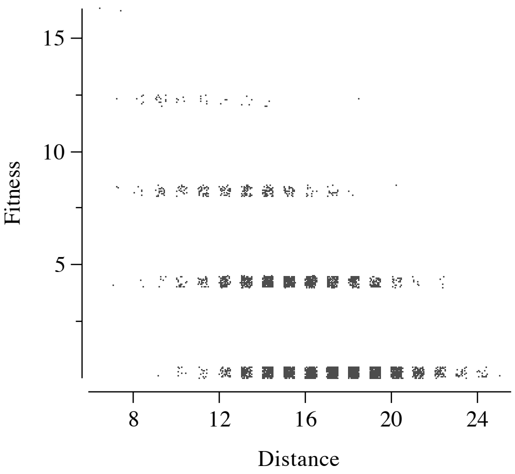


Figure 90. Royal road function R1 with 8 blocks of length 4 ($r = -0{\cdot}52$, 4000 sampled points).

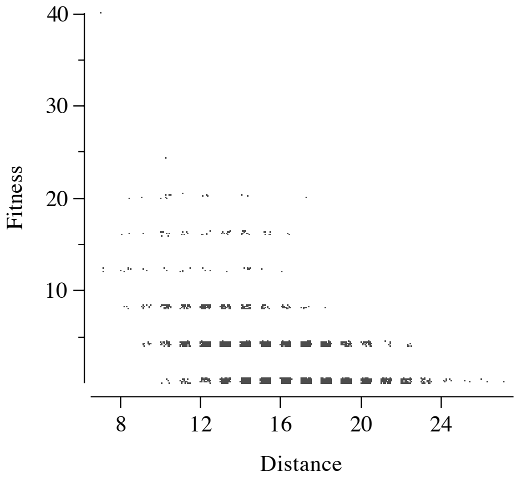


Figure 91. Royal road function R2 with 8 blocks of length 4 ($r = -0{\cdot}50$, 4000 sampled points).

5.7.1.3. Confirmation of Knowledge Regarding Coding

This section examines the effects of another choice that affects landscape structure that can also be detected by FDC, the influence of changes in Γ, the mapping from objects to representation. FDC's predictions regarding Gray versus binary coding (which is a change in Γ) lead to the discovery that the superiority of one code over another depends on the number of bits used to encode the numeric values.

Gray Versus Binary Coding

Although it is common knowledge that Gray coding is often useful to a GA in function optimization, there has never been any method of deciding whether one coding will perform better than another, other than simply trying the two. Experiments along this line have been performed by Caruana and Schaffer [159]. They studied De Jong's functions [30] (and one other) and found that Gray coding was significantly better than binary coding, when considering online performance, on 4 of the 5 functions. When they looked at the best solutions found under either coding, Gray was significantly better on one of the five. In no case did binary coding perform significantly better than Gray.

Using FDC, it is possible to examine different encodings and make predictions about which ones will be better. Surprisingly, FDC's predictions about the relative worth of binary and Gray codes depended on the number of bits in the encoding.

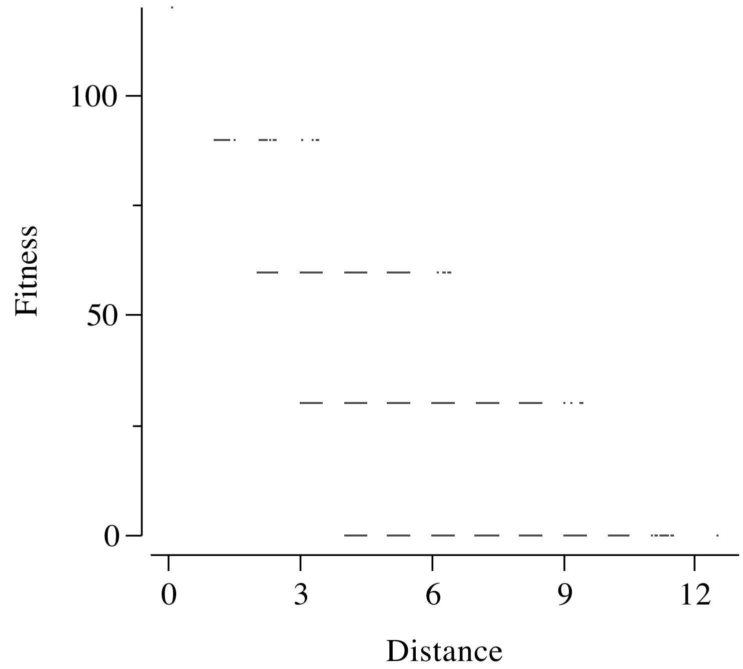


Figure 92. Ackley's plateau function on 12 bits ($r = -0{\cdot}65$).

Experiments with a GA have indicated that these predictions are accurate. For example, consider the positions in Figure 53 of F2(n) and GF2(n). F2 is a problem on two real variables, so F2(n) indicates that $n/2$ bits were used to code for each variable. When $n = 8$, we calculated $r(F2(8)) = -0{\cdot}24$ whereas $r(GF2(8)) = -0{\cdot}06$, indicating that with 8 bits, binary coding is likely to make search easier than Gray coding.[1] Figures 93 and 94 show a clear difference in the encoding on 8 bits. But now consider $r(F2(12)) = -0{\cdot}10$ versus $r(GF2(12)) = -0{\cdot}41$. With 12 bits, Gray coding should be better than binary. The scatter plots for F2(12) and GF2(12) are shown in Figures 95 and 96. When we move to 16 bits, we get $r(F2(16)) = r(GF2(16)) = -0{\cdot}19$ (Figures 97 and 98). Finally, with 24 bits (the number used by Caruana and Schaffer), we have $r(F2(24)) = -0{\cdot}09$ and $r(GF2(24)) = -0{\cdot}26$ (Figures 99 and 100). Once again, Gray coding should be better (as was found by Caruana and Schaffer when considering online performance).

[1] As the De Jong functions are minimization problems, positive r values are ideal. We have inverted the sign of r for these functions, to be consistent with the rest of the dissertation. It is straightforward to prove that this is the correlation that will obtain if we convert the problem to a maximization problem via subtracting all original fitnesses from a constant.

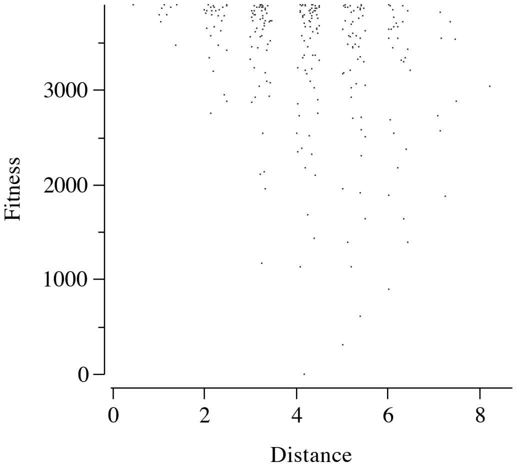


Figure 93. De Jong's F2 binary coded with 8 bits converted to a maximization problem ($r = -0{\cdot}24$).

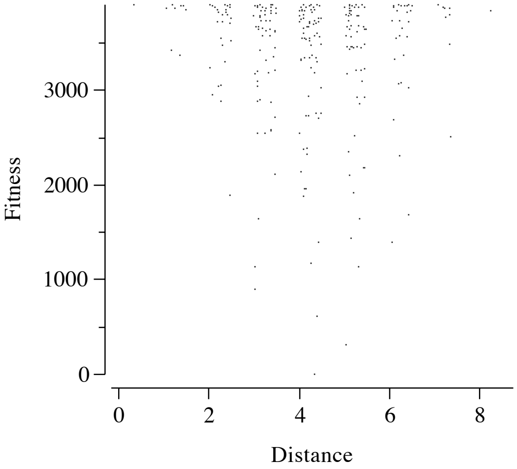


Figure 94. De Jong's F2 Gray coded with 8 bits converted to a maximization problem ($r = -0{\cdot}06$).

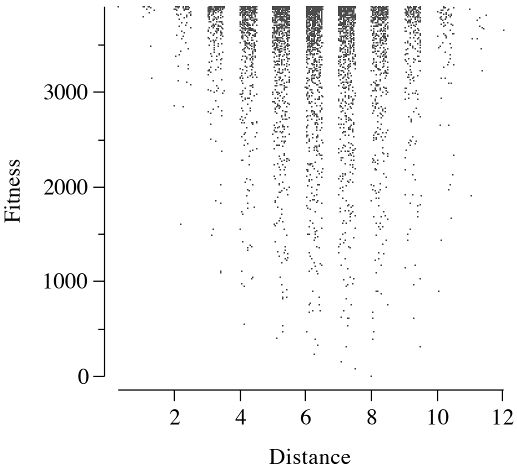


Figure 95. De Jong's F2 binary coded with 12 bits converted to a maximization problem ($r = -0{\cdot}10$). The unusual appearance is due to the presence of many highly fit points and "cliffs" in the encoding.

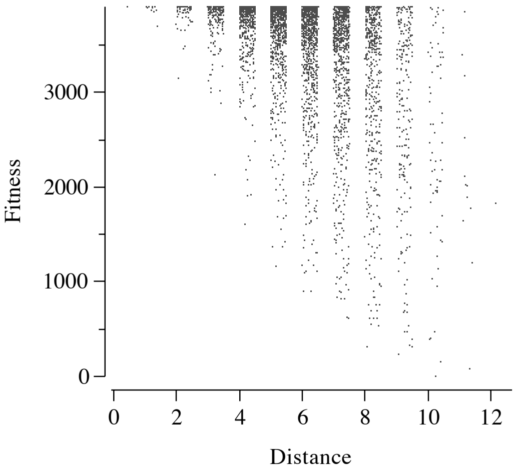


Figure 96. De Jong's F2 Gray coded with 12 bits converted to a maximization problem ($r = -0{\cdot}41$).

Are the reversals in FDC reliable indicators of GA performance? Preliminary results indicate that the answer is yes. For example, in 10,000 runs of a fairly standard GA (population 50, 25 generations, tournament selection, mutation probability 0·001, two-point crossover probability 0·7), binary coding on 8 bits finds the optimum approximately 4% more often than Gray coding does. On 12 bits, the GA with Gray coding finds the optimum approximately 4% more often than it does with binary coding. On 16 bits, the difference between the two falls to approximately 0·5%. This reversal from 8 to 12 bits and then equality at 16 bits are exactly what FDC predicts.

It is difficult to predict the results of Caruana and Schaffer for a number of reasons. One difficulty is that FDC is telling us something about how difficult it is to locate global maxima, whereas Caruana and Schaffer examined online performance and best fitness. While these are obviously related to the FDC measure, it is not clear to what extent FDC should be a reliable predictor of these performance criteria. As another example of this difficulty, FDC indicates that Gray coding will perform worse than binary coding on F3 when 15, 30 and 50 bits are used, yet Caruana and Schaffer found no significant difference between binary and Gray on this function for either of their performance measures. Does this mean that FDC has erred? Not necessarily. The situation is resolved when one notices that the resources given to the GA by Caruana and Schaffer were enough to allow their GA to solve the problem on every run under both encodings. There does appear to be a difference for a GA on this problem, but it can only be illustrated when the GA is given limited resources. For instance, on 15 bits, with a population of size 50 and 25 generations mutation probability 0·001 and two-point crossover probability 0·7, a standard GA found the optimum 1050 times out of 2000, compared to only 673 out of 2000 using Gray coding. With 30 bits, a population of 100 and 50 generations, binary coding succeeded 100 out of 500 runs, while Gray coding succeeded only 34 out of 500. Figures 101 and 102 show sampled scatter plots for 15 bits.

Problems such as these make it difficult to compare FDC with the results of Caruana and Schaffer. A comparison with the binary coding results of Davis [160] is even more difficult since we do not know how many bits were used to encode variables, which, as we have seen may alter performance considerably, and the evaluation metric used by Davis was unusual. At this stage we have found nothing to indicate that FDC is misleading in its predictions regarding Gray and binary coding. Our preliminary experiments with a GA, using the number of times the global optimum is encountered as a performance measure, have all matched FDC's predictions.

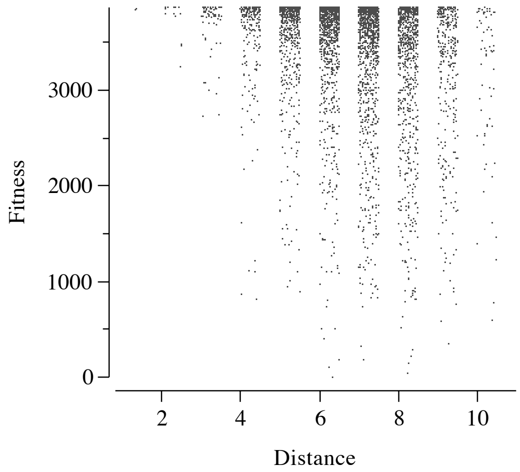


Figure 97. De Jong's F2 binary coded with 16 bits converted to a maximization problem ($r = -0{\cdot}18$). encoding.

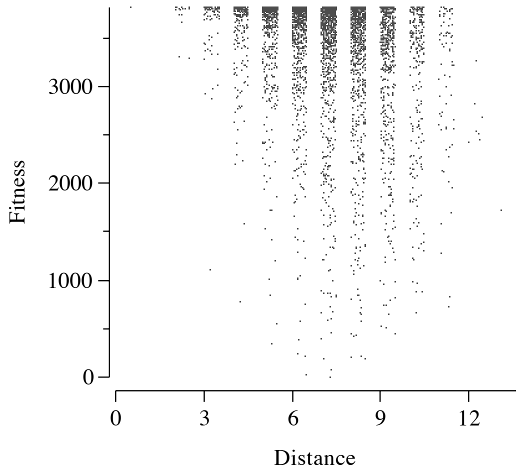


Figure 98. De Jong's F2 Gray coded with 16 bits converted to a maximization problem ($r = -0{\cdot}21$).

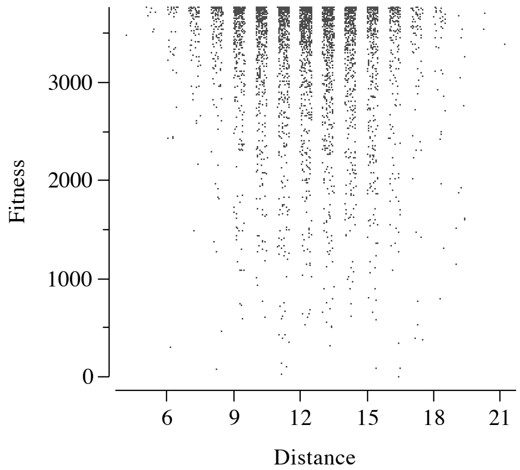


Figure 99. De Jong's F2 binary coded with 24 bits converted to a maximization problem ($r = -0{\cdot}09$).

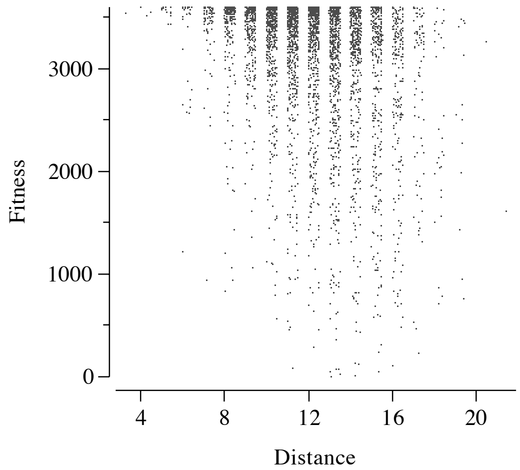


Figure 100. De Jong's F2 Gray coded with 24 bits converted to a maximization problem ($r = -0{\cdot}26$).

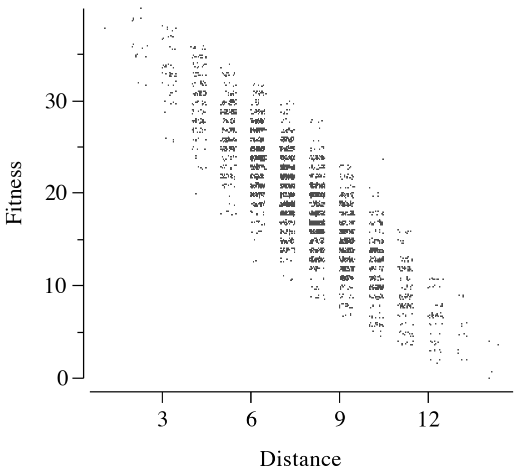


Figure 101. De Jong's F3 binary coded with 15 bits converted to a maximization problem ($r = -0{\cdot}86$, 4000 sampled points).

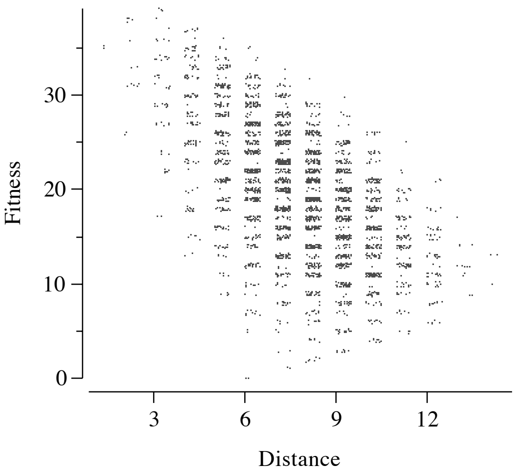


Figure 102. De Jong's F3 Gray coded with 15 bits ($r = -0{\cdot}57$, 4000 sampled points).

Liepins and Vose's Transform

The deceptive problem of Liepins and Vose [47] exhibits almost perfect correlation ($r = 0{\cdot}99$), as can be seen in Figure 105. The transformation they give alters what they term the "embedding" of the problem, which is denoted by Γ in the model of this dissertation. This transformation is simply another way to interpret bit strings in the algorithm as corresponding to objects that are potential solutions to the search problem. Binary coding and Gray coding are also mappings of this kind. The transformed problem is interesting because it is described as "fully easy" yet it has almost zero correlation. Examining scatter plots of fitness and distance gives the explanation; the plots resemble an X, as shown in the example on ten bits in Figure 106. Approximately one half of the space has a very good FDC ($r \approx -1$) and the other half has a poor FDC ($r \approx 1$). The overall result is an r value close to zero. This is the second example of a way in which correlation can prove a poor summary statistic. Once again, the structure in the problem is quickly revealed when the scatter diagram is plotted, reinforcing our claim that it is the relationship between fitness and distance which is important and that important aspects of this may be missed if one relies solely on correlation. It is reasonable to expect that the portion of the space with FDC approximately -1 should allow the global solution to be found without undue trouble. A standard GA (population 100, 50 generations, mutation probability 0·001, two-point crossover probability 0·7 and binary tournament selection) found the

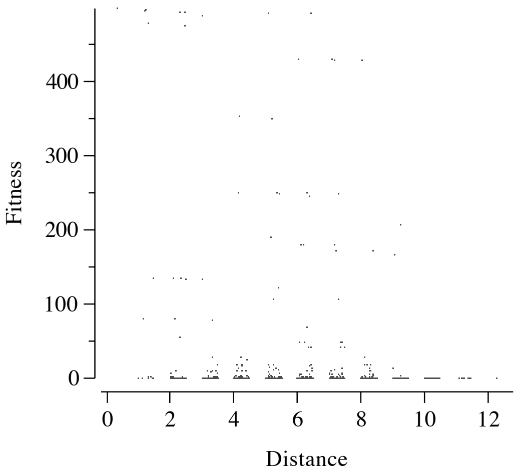


Figure 103. De Jong's F5 binary coded with 12 bits converted to a maximization problem ($r = -0{\cdot}86$).

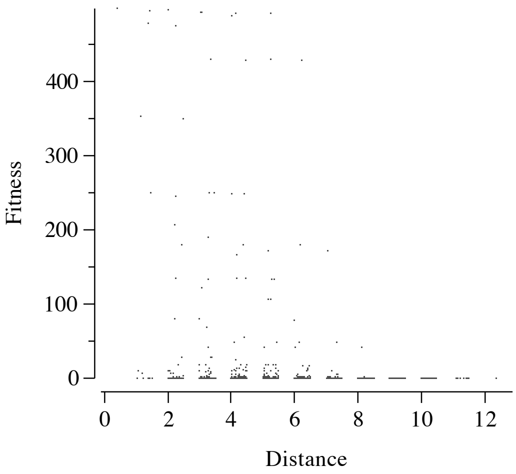


Figure 104. De Jong's F5 Gray coded with 12 bits ($r = -0{\cdot}57$).

global optimum 59 times out of 500.

5.7.2. Discussion

There is an intuitively appealing informal argument that the correlation between fitness and distance is what is important for success in search. Suppose you get out of bed in the night, hoping to make your way through the dark house to the chocolate in the fridge in the kitchen. The degree to which you will be successful will depend on how accurately your *idea* of where you are corresponds to where you *actually* are. If you believe you're in the hallway leading to the kitchen but are actually in the bedroom closet, the search is unlikely to end happily. This scenario is also the basis of an argument against the claim that good parent/offspring fitness correlation is what is important for successful search. Determining whether the floor is more or less flat will not help you find the kitchen. If r has a large magnitude, we conjecture that parent/child fitness correlation will also be high. This is based on the simple observation that if FDC is high, then good correlation of fitnesses between neighbors should be a consequence. Thus good parent/child fitness correlation is seen as a necessary but not sufficient condition for a landscape to be easily searchable. If this conjecture is correct, such correlation is not sufficient as it will also exist when FDC gives a value that is large and positive. This will likely be unimportant in real world problems, if problems with large positive FDC are purely artificial constructions of

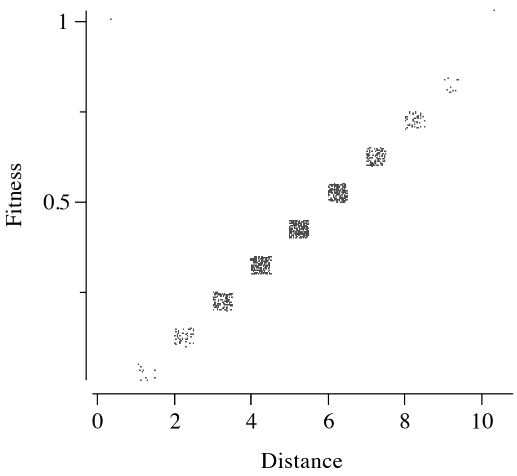


Figure 105. Liepins and Vose's fully deceptive problem on 10 bits ($r = 0{\cdot}98$).

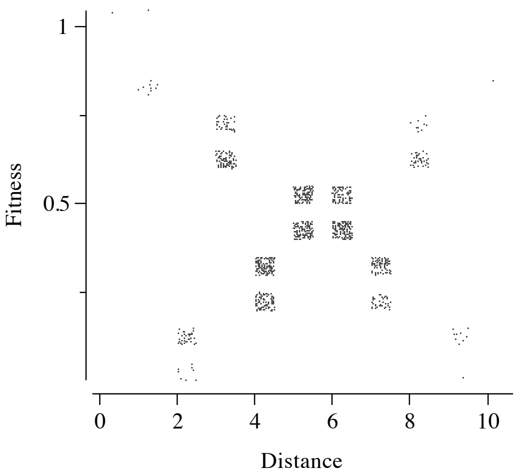


Figure 106. The transform of Liepins and Vose's fully deceptive problem on 10 bits ($r = -0{\cdot}02$). Correlation cannot detect the X structure.

the GA community.

It is far from clear what it means for a problem to be difficult or easy. As a result it is inherently difficult to test a measure of problem difficulty. A convincing demonstration will need to account for variability in the resources that are used to attack a problem, the size of the problem, variance in stochastic algorithms, and other thorny issues. FDC will only give an indication of how hard it is to locate what you tell it you are interested in locating. If you only tell it about global maxima, it is unreasonable to expect information about whether a search algorithm will find other points or regions. If all the global optima are not known and FDC is run on a subset of them, its results may indicate that the correlation is zero. When the other optima are added, the correlation may be far from zero. FDC is useful in saying something about problems whose solutions are already known. It can be hoped that information on small examples of problems will be applicable to larger instances, but in general this will not be the case. FDC is intended to be a general indicator of search difficulty, and is not specific to any search algorithm. In particular, FDC knows nothing whatsoever about a GA. This is both encouraging and alarming. It is encouraging that the measure works well on a large number of problems, but is alarming since, if we are to use it as a serious measure of problem hardness for a GA, it should know something about the GA. Probably the best interpretation of FDC values is as an indication of approximately how difficult a problem *should* be.

For example, if $r = -0{\cdot}5$ for some problem, but a GA never solves it, there is an indication that the particular GA is doing something wrong.

Hamming distance is *not* a distance metric that applies to any of a GA's operators. Distance between strings s_1 and s_2 under normal GA mutation is more akin to the reciprocal of the probability that s_1 will be converted to s_2 in a single application of the mutation operator. Naturally, Hamming distance is strongly related to this distance and this is presumably one reason why FDC's indications correlate well with GA performance. A more accurate measure might be developed that was based on the distances between points *according to the operator in use by the algorithm.* That Hamming distance works at all as an indicator of GA performance is a strong hint that a simplistic (i.e., easily computed) distance metric on permutations (for example, the minimum number of remove-and-reinsert operations between two permutations) may also prove very useful as a metric in FDC when considering ordering problems, *even if the algorithms in question do not make use of that operator.* A similar approach, also successful, was been adopted by Boese, Kahng and Muddu [116] (see §6.4(169) for some details).

Because the computation of FDC relies on prior knowledge of global optima, the measure does not appear well-suited for prediction of problem difficulty. Ongoing research is investigating an approach to prediction, based on the relationship between fitness and distance. In this approach, an apparently high peak is located (via some number of hillclimbs) and FDC is computed *as though* this peak were the global optimum.[2] This gives an indication of how hard it is to find that peak. This process is repeated several times to get an overall indication of the difficulty of searching on the given landscape. It is expected that such a predictive measure could be "fooled" in the same way that measures of correlation length can be fooled. That is, it will classify a deceptive problem as easy because, if the single global optimum is ignored, the problem *is* easy. In practice, it is probably more desirable that a measure of problem difficulty reports that such problems are easy. Only a very strict and theoretical orientation insists that these problems be considered difficult.

Discussion in this section has concentrated mainly on fitness and distance. It should be noted that FDC is affected by not only fitness function, but by operator (which gives distance), by representation and by Γ (the mapping from objects in the real world to the representation used by the search algorithm). Because these will all affect FDC, it follows that FDC should be useful in comparing all of these choices.

[2] It is not even necessary for the vertex from which distance is computed to be a peak. A randomly chosen vertex will still allow the calculation of FDC.

5.7.3. Conclusion

This chapter presented a perspective from which evolutionary algorithms may be regarded as heuristic state space search algorithms. Much of the change in perspective can be accomplished by a simple change in language that sets aside the biological metaphor usually employed when describing evolutionary algorithms. One aspect of the correspondence between the fields was examined in detail, the relationship between fitness and heuristic functions.

The relationship between fitness and distance to goal has a great influence on search difficulty for a GA. One simple measure of this relationship is the correlation coefficient between fitness and distance (FDC) which has proved a reliable, though not infallible, indicator of GA performance on a wide range of problems. On occasion, correlation is too simplistic a summary statistic, in which case a scatter plot of fitness versus distance will often reveal the structure of the relationship between fitness and distance. FDC can be used to compare different approaches to solving a problem. For instance, FDC predicted that the relative superiority of binary and Gray coding for a GA was dependent on the number of bits used to encode variables. Subsequent empirical tests have supported this.

The development of FDC proceeded directly from thinking in terms of a model of search which views the GA as navigating on a set of landscape graphs. AI has long regarded search from a similar perspective, and a simple change in language is sufficient to view GAs as state-space search algorithms using heuristic evaluation functions. In AI, the heuristic function is explicitly chosen to be as well correlated with distance to the goal state as possible, and it is easy to argue that a similar fitness function in a GA will make for easy search. Having done that, it is a small step to consider to what extent our current GA landscapes match this ideal, and to use that as an indicator of search difficulty. The fact that this proves successful is likely to be unsurprising to those in the AI community who work on heuristic search algorithms. I believe there is much that can be learned about GAs via considering their relationship with heuristic state-space search.

CHAPTER 6

Related Work

6.1. Introduction

This chapter divides work related to this dissertation into three areas (1) work on landscapes in other fields, (2) work on landscapes in computer science and evolutionary computation, and (3) work related to the connection between landscapes and heuristic search, especially to fitness distance correlation. Several pieces of research discussed below could be placed in more than one of these categories. This is merely one way of roughly grouping related work.

6.2. Related Work on Landscapes

The landscape metaphor originated with the work of Sewall Wright [16]. The idea has received wide attention within biology, but has also been adopted by researchers in other fields. This section provides an outline of the use of landscapes in biology, chemistry, computer science and physics.

6.2.1. Landscapes in Biology

Provine [87] has done an excellent job of summarizing the uses (and abuses) of Wright's "surfaces of selective value." This section merely touches on some of the points that are so well made by Provine. Wright's original diagrams were published in 1932 [16] and were intended to make it simpler to understand his earlier and far more mathematically sophisticated paper on population genetics [161]. The diagrams used by Wright became a popular vehicle for explanations of various phenomena in biology and have become known as "fitness landscapes." Unfortunately, they were widely used in at least three ways:

- In Wright's 1932 paper, each axis of his landscape corresponded to a gene combination. An additional dimension was added for fitness. Thus a point on a landscape corresponded to an individual gene combination and there was one axis for every possible combination. Because each "axis" corresponds to the genetic makeup of an individual, it is not clear how to interpret them as axes. The axes have no units and no gradations. Here is Provine's summary of Wright's original diagrams:

  > "Thus the famous diagrams of Wright's 1932 paper, certainly the most popular of all graphic representations of evolutionary biology in the twentieth century, are meaningless in any precise sense."
  >
  > Provine [87, page 310].

  In defense, Wright [37] acknowledges that his two-dimensional diagrams are "useless for mathematical purposes," but argues that they were never intended for such use. Rather, the diagrams were intended as a convenient pictorial representation of a process taking place in a high-dimensional space.

  Although these first published landscapes make little sense mathematically, the idea of the landscape above fits into the landscape model of this dissertation. Each genetic combination is represented as a vertex in a landscape graph and each is assigned a fitness that is used to label the vertex. The advantage of the new model is that it requires no notion of dimensionality, and can therefore be used naturally to produce a landscape graph that is perfectly well-defined.

- In a second interpretation of his landscapes, Wright regarded each axis as corresponding to the frequency of a give gene. Thus the landscape had as many axis as there were genes, and each axis went from zero to one according to the frequency of the gene in a population. A point in this space therefore corresponds to the gene frequencies of a population of organisms and fitness, the extra dimension, is given by the mean fitness of this population. Many populations might have the same gene frequencies however, and this leads to difficulties with this interpretation. This interpretation predates the one published in 1932. Wright tended to use this version mathematically, while continuing to publish the 1932 diagrams.

  The landscape model of this dissertation can be used to describe this form of Wright's landscapes if one views the process that creates one population from another as an operator that produces edges in a graph whose vertices correspond to entire populations. More in line with Wright's formulation would be a landscape in which a vertex corresponds to the infinite set of populations that exhibit a given distribution of gene frequencies. As Provine observes, this greatly complicates matters when one tries to assign a "fitness" to a vertex, [87].

- Simpson [19] was responsible for the introduction of a third interpretation in which axes measured structural (i.e., phenotypic) variation in individuals. In this version it appears that Simpson intended points in the space to correspond to individuals, with the extra dimension given by individual fitness. This landscape can be viewed as a graph in the model of this dissertation in an analogous fashion to that in which Wright's first version is described above.

These conflicting interpretations of fitness landscapes suggest that the metaphor was an attractive one to many. The lack of a rigorous definition did not prevent use of the landscape metaphor from becoming widespread. Actually, lack of definition may have encouraged this. There is nothing inherently wrong with this situation if landscapes are used to convey a vague picture of some process. When attempts are made to use landscapes in a rigorous fashion, it becomes more important to know what is being referred to. A similar situation exists in evolutionary computation, where the word landscape tends to be used in a rather cavalier fashion. Landscapes are frequently heard of but infrequently defined. In some instances this is very useful. In other cases, landscapes are used as the foundation of seemingly plausible arguments that, when closely examined, can be extremely difficult to make any sense of at all.

6.2.2. Landscapes in Physics and Chemistry

Two fields in which landscapes are used that do not suffer from a lack of formal definition are physics and chemistry. Formal definitions can be found in the work of Weinberger [63, 64] and in a number of papers by Fontana, Huynen, Schuster, Stadler and others in theoretical biology and chemistry (see [162, 163], and references below, for examples). Weinberger's landscapes are described in the following section. Both these models were formulated to study specific systems; the dynamics of spin glasses and the folding and evolution of RNA sequences and secondary structure. They are clearly defined and are useful for the study of these systems. However, not surprisingly, they are not directly applicable to evolutionary algorithms. There are three primary reasons for this:

- These models are only relevant to systems in which an operator acts on a single "individual" to produce another "individual." That is, one RNA sequence is converted to another, or one Hamiltonian circuit in a graph is converted into another.

- The systems under consideration all involve a single operator. For example, an operator which changes a spin in a spin glass or a point mutation operator which changes a nucleotide in an RNA molecule.

- All possible outcomes from an application of an operator have equal probability of occurrence.

These differences make the application of these landscape models to evolutionary algorithms problematic. In evolutionary algorithms, none of the above are true. These algorithms have operators that act on and produce multiple individuals, they employ multiple operators, and the possible results of operator application are not equally probable. These possibilities are incorporated in the landscapes model of this dissertation.

6.2.3. Landscapes in Computer Science

Weinberger's work has had a significant influence on work done in computer science [63, 64, 65, 164]. Weinberger provides a precise definition of a fitness landscape. His landscape is also a graph (though finite), whose vertices are labeled using some real-valued function. He introduces the autocorrelation function and shows its use as a measure of landscape fitness correlation or "ruggedness." This measure and others derived from this work have been relatively widely used [60, 61, 62, 70, 117, 118, 165]. The model presented in this dissertation generalizes Weinberger's model. It allows operators that act on and produce multisets of points in $\mathcal{R}$, allows the graph to be infinite and/or unconnected, and assigns probabilities to edges which is important when considering mutation or crossover. Figure 1 on page 31 illustrates this common situation with crossover. Weinberger considered operators that would assign an equal probability to each outgoing edge from a vertex, and so had no need for probabilities.

Manderick, De Weger and Spiessens [60] used Weinberger's landscape and autocorrelation function to examine the relationship between the statistical structure of landscapes and the performance of genetic algorithms on the same landscapes. This important paper illustrates how successfully statistics concerning landscape structure can predict algorithm performance. Manderick et al. viewed a landscape as something defined by an operator that acted on and produced a single point of $\mathcal{R}$. Weinberger's autocorrelation function can be calculated for these simplest of operators. They recognized crossover as an operator without making the generalization of the model of this dissertation. To deal with this, they defined a measure of operator correlation which they applied successfully to one-point crossover on NK landscapes and to four crossover operators for TSP. This statistic is calculated by repeated use of the operator from randomly chosen starting points. Thus, they were actually computing statistical measures about the crossover landscape without recognizing it as a landscape. Had they done so, they might have used the autocorrelation function, since they dealt with the walkable landscapes generated by crossover operators that take two parents and produce two children.

Taking the work of Manderick et al. as a starting point, Mathias and Whitley [61] examined other crossover operators for TSP and Dzubera and Whitley develop measures of "part" and "partial" correlation which they also apply to TSP. Much of the work mentioned above in theoretical chemistry has also been influenced by Weinberger's work (see, for example, [62, 166, 167, 168]). An overview of uses of the landscape perspective is far beyond the scope of this section. These works are mentioned to show the extent of the influence of Weinberger's model and definition of the autocorrelation function and correlation length.

The work of Nix and Vose and others on Markov chain analysis of GAs [52, 169, 170, 171, 172, 115] can also be looked at from a landscape perspective. As mentioned in §2.3(23), the choice of what one regards as an operator is a matter of perspective. Different choices will impact the ease with which we can study different aspects of search algorithms. By viewing an entire generation of a GA as an operator that converts one population to another, the GA can be imagined as taking a walk on a landscape graph whose vertices correspond to populations. This operator, like any other, can also be viewed as defining a transition matrix for a Markov chain. Naturally, for a process as complex as an entire GA generation, calculating the transition probabilities will be quite involved, but this is what Nix and Vose have done. Because this operator is used repeatedly (unlike simpler operators such as crossover and mutation which are used in series), the expected behavior of a GA can be determined via Markovian analysis. Due to the exponential growth in the number of possible states of this Markov process, such analysis is typically restricted to small populations consisting of short individuals. Nevertheless, it offers exact results and insights that are otherwise not available. Markov chain-like analysis of individual operators has been carried out by Goldberg and Segrest [173], Mahfoud [174], Horn [175] and Whitley and Yoo [176].

Recently, Culberson [29] independently conceived of a crossover landscape. His structure, which he calls a *search space structure*, is also a graph and the vertices correspond to a population of points from $\{0,1\}^n$. He examines crossovers between two complementary strings which creates a graph that corresponds to the largest connected component of the crossover landscape generated by $\chi_1^{2\to2}$ (two parents, two offspring, one crossover point) in the current model. That component (like all others) is a hypercube. It contains vertices that correspond to all possible pairs of binary strings of the form $(a, \overline{a})$. Culberson shows that the structure of the component is isomorphic to the hypercube generated by the bit-flipping operator for strings of length $n-1$. He shows the existence of $\chi_1^{2\to2}$-local-minima in this structure and demonstrates how to transform a problem that appears hard for one operator into a problem that appears hard for the other. This provides further evidence of the importance of structure for search and of how that structure is induced by the choice of operator. Wagner is also investigating isomorphisms between crossover and mutation landscapes [177].

There is other related work whose connections to the landscape model have not yet been well explored. This includes work done by Whitley and by Vose on mixing matrices [178, 179], work by Altenberg [180], whose transmission functions for crossover operators in $\phi_{2\to1}$ can be drawn as landscape graphs similar to that shown in Figure 11 on page 54, and Helman's work on a general algebra for search problems [140].

6.3. Work Related to the Heuristic Search Connection

Tackett [119, 120] proposes that there is a connection between Genetic Programming (and thus GAs) and heuristic search. In particular, he outlines a correspondence between GP and beam search [181]. The correspondence is based on the fact that the population in a GA can be viewed as akin to the priority queue of beam search. Population members in a GA are "expanded" (chosen to produce offspring) in a fashion prioritized by fitness. In beam search, nodes (individuals) are expanded strictly according to position in the priority queue. This correspondence fits perfectly into the correspondence between evolutionary algorithms and heuristic search developed in Chapter 5 using the language changes summarized in Table 31 on page 125. Tackett [119] also makes the point that the uniform use of biological language to describe GAs tends to obscure the connection with heuristic search.

6.4. Work Related to FDC

There are four pieces of work (that I am aware of) which present diagrams similar to those used to compute fitness distance correlation. In each case, the diagrams are closely related to the scatter plots used for FDC, but the uses to which they are put (if any) and the research they motivate are all quite different.

1. Kauffman ([117, page 564] and [118, page 61]) shows scatter plots of the fitness of local minima versus their distance from the best local minimum found, for several NK landscapes. A number of local minima are discovered through hillclimbing, and each is represented by a single point in the plot. These diagrams demonstrate the existence of what Kauffman calls a "Massif Central" in NK landscapes with small K. By this it is meant that the highest peaks in the landscape are near each other and this region occupies a very small region of the entire search space. These scatter plots differ from the plots used in FDC only in the fact that each point in the plots corresponds to a peak on the landscape, not to a randomly chosen vertex in the landscape. Kauffman does not compute correlation or other statistics about the relationship between distance from (apparent) global optimum and fitness of local optima.

2. Boese, Kahng and Muddu [116] produce similar scatter plots, but on a variety of different problems, and use them to motivate the construction of "multi-start" hillclimbers that take advantage of the "big valley" structure they observe. Their "big valley" is Kauffman's "Massif Central" inverted (Boese et al. study minimization problems). They observe that the highest peaks (found by various hillclimbers based on different heuristics) are tightly clustered. They use this knowledge to construct an "adaptive multi-start" hillclimber that takes uphill steps in the same way the BH algorithm of §3.6(55) does. The hillclimber uses the 2-*opt* operator introduced by Lin and Kernighan [182]. The algorithm assumes that the problem at hand exhibits the big valley structure, and they show how the hillclimber compares very favorably with other hillclimbers that do not take advantage of this global structure. The hillclimber uses information about high fitness peaks located earlier in the search to select new starting vertices on subsequent climbs. This technique is applied to 100- and 500-city randomly generated symmetric traveling salesperson problems and to 100- and 150-vertex graph bisection problems. Boese [183] extends this work to consider the 532-city traveling salesperson problem of Padberg and Rinaldi [184] and

mentions that similar plots have been obtained on circuit partitioning, graph partitioning, number partitioning, satisfiability, and job shop scheduling.

In addition to scatter plots showing fitness of local optima against distance to global optimum, Boese et al. produced plots of fitness of local optima versus mean distance to other local optima. On the problems they have studied, there has (apparently) been a single global optimum, rather than many, as with some problems studied with FDC (which uses distance to the nearest global optimum). Boese [183] produced similar plots on the 532-city problem using local optima located via several hillclimbers, each using a different "heuristic" ("operator" in the language of this dissertation). The results for these different hillclimbers allow Boese to compute the size of the "big valley" for each operator. The correlation between fitness of local optima and distance to global optimum was also computed and used to examine the relative strengths of the relationship for each of the operators.

Boese et al. use a distance metric that is not the same as the distance defined by the operators they employ. There is no known polynomial time algorithm for computing the minimum number of 2-*opt* moves between two circuits in a graph. Instead, they use "bond" distance which is the number of vertices in the graph minus the number of edges the two circuits have in common. Some analysis shows that this distance is always within a factor of two of the 2-*opt* distance between the circuits. This is similar to the use of Hamming distance instead of "mutation distance" or "crossover distance" in the calculation of FDC.

Because the scatter plots of Boese et al. examine local optima, the relationship that will be observed between fitness and distance will depend on the difficulty of the problem and on the quality of the local optima that are examined. If a good hillclimber is used to locate local optima on an easy problem, few local optima may be found. In the worst case, on a non-pathological (in the style of Horn et al. [86]) unimodal problem, only one peak will be located, and it will be the global optimum. Boese et al. and Hagen and Kahng [185] have observed instances where the relationship between fitness and distance deteriorates and this appears dependent not only on the problem at hand, but on the quality of the heuristic used to locate local optima. FDC does not run into problems in these situations because it considers randomly chosen vertices of the landscape—there is no requirement that they be local optima. Thus FDC is useful for classifying easy problems as well as difficult ones. This is of little consequence to Boese et al., who are motivated to take advantage of the big valley structure in problems that contain many local optima. Naturally, the pervasiveness of the "big valley" structure across problem types and instances will determine how useful the adaptive multi-start algorithm is. Judging from these initial investigations, the class of problems possessing this structural property may quite large. If so, algorithms designed to exploit this structure, such as adaptive multi-start, will be hard to beat.

3. Ruml, Ngo, Marks and Shieber have also produced a scatter plot of fitness against distance from a known optimum [186]. They examined an instance of the number partitioning problem (of size 100) and compared points in several representation spaces to the optimum obtained by Karmarkar and Karp [187] (KK-optimum). Ruml et al. generate points by taking directed walks away from the global optimum, at each step increasing the distance from the KK-optimum by one. The points found on several such walks are then plotted as fitness (normalized by the fitness of the KK-optimum) versus distance from the KK-optimum. The resulting plots exhibit a "bumpy funnel" in which there is a strong correspondence between solution quality and distance from the KK-optimum. As with the observations of Boese et al. [116, 183], the high quality solutions are tightly clustered around the best-known optimum and Ruml et al. suggest that search algorithms that take advantage of this type of structure (which they term "gravitation") will prove competitive. It is clear from these results that the problem structure around the KK-optimum is qualitatively like that around the optima studied by Boese et al.. Unlike Boese's results with the 532-city TSP problem, we cannot be so confident that the KK-optimum is the global optimum as Karmarkar and Karp's algorithm exhibits a strong bias towards equal-sized partitions. The relationship between fitness and distance is not quantified in any way—the scatter plots are used to give pictorial evidence of the structure of the problem and "gravitation."

4. As mentioned in §1.3(8), Korf presented several heuristics for the $2 \times 2 \times 2$ version of Rubik's cube [33]. Korf's graphs have heuristic value on the X axis and the mean distance of all cube configurations with a given heuristic value on the Y axis. At first glance, these graphs would seem to have little to do with the FDC scatter plots of Chapter 5, but this is not the case. If the axes are swapped and each point is replaced by the actual set of points that it represents, the result is a scatter plot. As Korf points out, there will be virtually no correlation between fitness and distance in any of these plots. This replacement and inversion of axes has not been done, but Korf's diagrams make it apparent

that if correlation is not zero, it can only be slightly negative (this is undesirable in Korf's heuristic functions, which must be maximized).

Finally, there is a relationship between FDC and Weinberger's correlation length [65]. Correlation length is a measure of correlation between fitness values, but the autocorrelation function, from which correlation length is calculated, also takes distance into account. The autocorrelation function provides the correlation between fitnesses at all distances. The connection between this measure and FDC is the subject of current investigation. In particular, if FDC proves useful for prediction using the method suggested in §5.7.2(160), it will likely be prone to "error" in the way that correlation length is when the fitness landscape is not isotropic. When FDC is computed from the point of view of a randomly chosen vertex (or peak), it, like correlation length, will indicate that fully deceptive problems (for example) are easy. In practice there may be nothing wrong with this, since if we ever encounter a reasonably large fully deceptive problem, it would be a simple problem, as far as we would ever be able to tell. The global maximum may as well not exist. A measure of problem difficulty that accords well with what we actually experience is likely to be far more useful than some measure which insists that such a problem is actually very hard.

CHAPTER 7

Conclusions

In the introduction to this work, I wrote that the dissertation was a collection of very simple questions and my attempts to answer them. It seems appropriate then that a concluding chapter should present these questions and summarize (1) the answers I have found, (2) the increase in understanding that the answers bring, and (3) the practical benefits that result from the understanding.

What is a landscape?

This is the most fundamental question of this dissertation. In the model of this dissertation, a landscape is a labeled, directed, graph. The vertices of the graph correspond to multisets of individuals and the edges correspond to the actions of an operator. If an operator acts on an individual A, producing individual B with probability p, then the landscape graph contains an edge from the vertex representing A to the vertex representing B and that edge is labeled with p. It is important to allow a vertex to correspond to a multiset of individuals, not just a single one. By so doing, landscape graphs are well defined for operators that act on and produce multiple individuals. The vertices of a landscape graph are labeled with a value that can be thought of as a height, and this gives rise to the landscape image. When vertices correspond to single individuals, they are typically labeled with values from a fitness function. If not, the label is some function of the fitnesses of the individuals represented by the vertex. This, in essence, is the landscape model of this dissertation.

When evolutionary algorithms are viewed from the perspective of this model, there is an important consequence. Because these algorithms are usually viewed as making use of several operators, they are not operating on a single landscape, but on many. As a result, there are mutation landscapes, crossover landscapes, selection landscapes—every operator creates a landscape graph. Every time an algorithm uses a particular operator, it can be seen as traversing an edge, or taking a "step," on the landscape defined by the operator in question. I call this the "One Operator, One Landscape" consequence of the current model. If we intend to study and talk about landscapes, my model insists that we pay attention to the "one operator, one landscape" consequence. Naturally, we can study the results of using several operators in concert, but we can ask simpler questions—questions about the individual landscapes.

There are other consequences of the model, and these are described in §2.7(36). Two of these are the fact that landscapes may not be connected, and that they may not be *walkable*.

The model of landscapes presented in this dissertation was designed to be useful for thinking about evolutionary algorithms. Other models, designed for different purposes, have limitations that make them inappropriate for this purpose. The difficulties are that these other models do not (1) allow for algorithms that employ multiple operators, (2) account for operator transition probabilities, or (3) admit operators that act on or produce multiple individuals.

What Makes a Search Algorithm?

The landscape model is a very general one, and its usefulness is not restricted to evolutionary algorithms. The model is presented in terms of search, which leads one to ask what are the components of a search algorithm. If we suppose that the task of a search algorithm is to find some object or objects, we can ask what kinds of objects are examined by the algorithm and what its methods are for generating new objects when looking for the solution. The landscape model regards the methods used for creating new objects to examine as operators. Each operator creates the edges in a landscape graph, the vertices of which correspond to the collections (or multisets) of objects that the operator acts on and produces.

But a search algorithm is more than a collection of landscape graphs. The algorithm must explore these graphs in search of a solution object. The landscape model tells us nothing about this aspect of search. From the point of view of the model, search algorithms can be divided into landscapes and everything else. I call the "everything else" the *navigation strategy* of the search algorithm. The navigation strategy determines how and in what order the graphs are explored, from where exploration should commence, how it should be altered as the search proceeds, and when it should be halted. Thus, search algorithms may be thought of as composed of structure(s) and strategy.

It is reassuring to note that this perspective, though arrived at through thinking about evolutionary algorithms and landscapes, is nothing more than the perspective on search that has long been held in the Artificial Intelligence community. For example, production systems have been described as consisting of a "database," "production rules," and a "control strategy." These correspond fairly well to our landscape view of search with its vertices, edges (operators), and navigation strategy. Establishing links with the Artificial Intelligence search community is a good reality check, and, as will be seen, these links are more than superficial.

Can the Pieces be Reassembled?

If search algorithms can be decomposed into landscape structures and strategies, can we take several search algorithms apart and reassemble them into new algorithms? The answer is that we can do this and that it is easy. The reason is that landscapes are graphs and navigation strategies are designed to search graphs. So long as the landscape graphs of one algorithm satisfy any assumptions made by the navigation strategy of another algorithm (e.g., that the graph is a tree or that it is connected), there seems to be no logical reason to prevent the combination of pieces of search algorithms in any way we choose. With nothing more than the perspective offered by the landscape model, a vast number of possible new algorithms suggest themselves. Hybrid search algorithms are nothing new, but the landscape model encourages this and gives a formal framework for doing it. The field of possibilities can be explored systematically, rather than through investigations that follow occasional creative leaps.

Though this perspective might be obvious to those who study search in other contexts, it suggests novel ideas for hybrid algorithms whose pieces are drawn from evolutionary algorithms. The model seamlessly accommodates operators that act on multiple individuals, and so there is no difficulty exploring the graphs that correspond to these operators. To anyone familiar with these algorithms, and with the important unanswered questions about them, an attractive first choice of targets is an algorithm that explores the crossover landscape of a genetic algorithm. The simplest (interesting) navigation strategy is some form of hillclimbing, and so is born the *crossover hillclimber*.

The crossover hillclimber uses a simple hillclimbing strategy to explore the landscape graph of a crossover operator (§3.7(57)). On virtually all of the problems studied in Chapter 3, the algorithm performs better than either the bit-flipping hillclimbing algorithm from which the navigation strategy was taken, or the genetic algorithm from which the landscape was taken. The division into structure and strategy can be made, the pieces can be combined in novel ways, and the resulting algorithms can be practical. The success of the crossover hillclimber leads to another simple question.

Is Crossover Useful?

This question has no simple answer because it is too general. It is tautological to answer that crossover is useful if it manages to orchestrate the exchange of building blocks between individuals. The question we should be asking is: “Given a particular set of choices about algorithm, representation and fitness function etc., does the use of crossover produce benefits that other, simpler, operators cannot produce?” If so, there is a good argument for the use of crossover in that situation. A more complex subsequent question could ask for the characteristics that these situations have in common. In other words, as Eshelman and Schaffer asked, what is crossover’s niche? [89].

From a cursory inspection of the crossover hillclimbing algorithm, it appears that the algorithm has isolated crossover from a genetic algorithm and harnessed its power by searching using a more exploitative navigation strategy. New individuals are either created completely at random or via the use of crossover. As it is easily shown that the randomly created individuals are of predictably low quality, the credit for success of the algorithm must go to crossover. This reasoning is correct, but it is far from being the whole story.

Closer examination of the crossover hillclimbing algorithm reveals that its performance correlates extremely well with the number of crossovers it performs in which one individual is randomly created (§3.10(72)). But crossovers involving one random individual are nothing more than macromutations of the non-random individual, where the mutations are distributed in the fashion of the crossover operator in question. The conclusion that crossover hillclimbing was successful because it was making macromutations was confirmed when an extreme version of the algorithm, in which every crossover involved one random individual, proved better than all previous versions of the algorithm. This leads to a distinction between the *idea* and the *mechanics* of crossover. The idea is what crossover is trying to achieve (the recombination of above average genetic material from parents of above average fitness) and the mechanics is the method by which this is attempted. All crossover operators share the same basic idea, but their mechanics vary considerably. The crossover hillclimbing algorithm is a clear demonstration that crossover can be used effectively for search even when it is only the mechanics that are doing the work.

Can the Usefulness of Crossover be Tested?

Crossover can be useful in two ways, one of which is purely macromutational. To decide whether crossover is more useful than other, simpler, operators therefore requires answering whether the idea of crossover is producing gains over what could be obtained through macromutation alone. The standard test for determining if crossover is useful cannot answer this question. The standard test involves comparing a genetic algorithm with crossover to one without crossover. This is a comparison between an algorithm with both the idea of crossover and the mechanics that attempt to implement it, and an algorithm with neither of these. If the genetic algorithm with crossover does better than the one with no crossover, the conclusion that crossover is therefore useful (i.e., that it is doing more to aid the search than a simple macromutational operator would) is not justified. Enter the headless chicken.

The headless chicken test, in which a genetic algorithm with crossover is compared to one with *random crossover* is more informative (§3.12(77)). In random crossover, two random individuals are generated and used in normal crossovers with the parents. One offspring from each of these crossovers is passed back to the genetic algorithm. There is no communication between members of the population. The random crossover operator throws out the idea of crossover while keeping the mechanics. If the genetic algorithm with crossover does not outperform the genetic algorithm with random crossover, then by our definition of usefulness, crossover is not useful. Crossover is not producing gains that could not be obtained with a simpler operator (in this case macromutation). If the genetic algorithm with crossover fails the test (i.e., it does not outperform the genetic algorithm with random crossover), we cannot conclude that the genetic algorithm with crossover is *not* combining building blocks. The test simply shows that the rate at which this is being done (if at all) is not sufficient to produce gains that are greater than those that can be obtained through simple macromutation.

When the genetic algorithm with crossover does not outperform the genetic algorithm with random crossover on a particular combination of representation, fitness function etc., it is difficult to argue that the genetic algorithm *in that configuration* is making effective use of the population. If the genetic algorithm with crossover is worse than the genetic algorithm with random crossover, the population may actually be *hindering* the search. Not only are building blocks not being combined at any significant rate, but the possibility of advancing the search via macromutations is reduced because a population is used. The random crossover algorithm has the benefit of being able to conduct wider exploration. It is not constrained to drawing partners for crossover from a partially converged population whose makeup varies relatively slowly over time.

If crossover is not useful, the usefulness of the population and therefore of the genetic algorithm is called into question. If the genetic algorithm does not pass the headless chicken test, it is either time to look for alternate representations, fitness functions and operators etc. for the genetic algorithm, or, it is time to look for a new algorithm.

What is a Basin Of Attraction?

One of the purposes of using the landscape metaphor is to discuss properties of landscapes. To some extent, landscapes can be studied independently of the search algorithms that will be used to explore them. While it is true that all the components of a landscape (vertices, edges, vertex labels and transition probabilities) are determined by choices that are part of a search algorithm, a particular set of choices may be made by many algorithms. Properties of a particular landscape will be relevant to the entire family of algorithms that make the set of choices that determine the landscape. Algorithms which make very similar choices will perhaps be operating on similar landscapes, and these landscapes can be compared.

A feature of landscapes that is often discussed are the basins of attraction of peaks. Given a landscape, the basin of attraction of a vertex v in landscape graph is the set of vertices that are connected to v by a path (directed towards v). As edges correspond to applications of an operator, the basin of attraction is the set of vertices from which a random walk, using the operator, could possibly reach v.

Natural questions about the basin of attraction $B_\phi(v)$ of a vertex v include the size of $B_\phi(v)$, the identity of the vertices in $B_\phi(v)$, the number of vertices $B_\phi(v)$ has in common with other basins, the probability that vertices in $B_\phi(v)$ will reach v, and the overall probability that v will be discovered via a random walk, using ϕ, from a randomly chosen starting vertex in the landscape. Many of these questions can be answered by the *reverse hillclimbing* algorithm developed in Chapter 4. Given a set of peaks on a landscape, reverse hillclimbing can discover their basins of attraction for a number of hillclimbers and use these results to indicate which of these has the highest probability of locating a peak in the desired set. An exact answer is provided (assuming the number of vertices in the basins are not so great as to exhaust available computational resources). Using methods based on (non-exhaustive) sampling to obtain this information is computationally prohibitive and the answers are necessarily inexact. Chapter 4 and the results in [110] show how the reverse hillclimbing method can be used to reach rapid and accurate conclusions about hillclimbers based on data about basins of attraction.

What Makes Search Hard?

This is the most important question of this dissertation. This question is extremely hard to answer in general, and it is made harder by the fact that hardness is in the eye of the beholder. If one algorithm finds a problem hard but another does not, is the problem hard? If one algorithm never solves a problem with a given level of computational resources, but always solves the problem when the resources available are doubled, is the problem hard? The theory of NP-completeness [141, 188, 189] has produced many beautiful results concerned with problem difficulty, but the fact that a problem *type* is NP-complete does not necessarily tell us anything about a given *instance* of that problem [190]. The Halting Problem is formally undecidable, but faced with a particular Turing machine, it may be trivial to determine whether it halts. Additionally, we may not be concerned with finding the optimal solution to a problem. We may be satisfied by solutions that are very far from optimal. How hard are these problems?

If, instead, we ask "What properties of a landscape make it hard to search?" we can make significant progress. By ignoring algorithms (apart from the choices they make that determine landscapes), the problem becomes at once easier and harder. We may decide that a certain landscape feature, for instance "ruggedness," makes search difficult. If, in some quantitative sense, a landscape can be classed as "rugged" (and therefore difficult), does that mean that all search algorithms that operate on that landscape will have difficulty? No, because search algorithms are more than landscapes; they also have navigation strategies. What is difficult for one navigation strategy may be simple for another. On the other hand, our task is simpler as we are ignoring navigation strategies and it should be possible to develop statistical measurements of landscape graphs that might correlate well with observed algorithm difficulty.

My answer to this question can be arrived at most simply by considering what properties a landscape might have that would make search *easy*. The vertices of any connected graph can be labeled with the length of the shortest path to a designated vertex (whose label is therefore zero). If the designated vertex corresponds to the optimal solution to a search problem the landscape becomes unimodal. If a function of this nature could be found for any instance of a problem, it could be used to solve SATISFIABILITY in polynomial time. We obviously cannot hope to find such a (fitness) function, but it might be reasonable to hope that the more closely our fitness function and landscape approximate this ideal, the easier search will be on the landscape.

A measure of this, Fitness Distance Correlation (FDC), is defined in Chapter 5. The application of FDC to approximately 20 problems from the literature on genetic algorithms shows it to be a very reliable measure of difficulty for the genetic algorithm. Correlation is only one summary of the relationship between fitness and distance from the goal of the search, and in some cases it is too simplistic. Scatter plots of sampled vertices on a landscape often reveal structure not detected by correlation. These plots are one way of looking for structure in a landscape graph and they are often surprisingly revealing. Apart from correctly classifying many problems correctly, including ones that gave surprising results when first studied. FDC also predicted that the question of whether or not Gray coding was beneficial to a genetic algorithm was dependent on the number of bits used in the encoding. The accuracy of this prediction was later confirmed experimentally.

FDC captures a property of a landscape graph. It has nothing to do with any search algorithm beyond the choices that determine the landscape. As a result, FDC can indicate that a problem should be easy, but an algorithm, because of its navigation strategy, may find the problem hard. The indications of FDC about a problem can be interpreted as a rough measure of how hard a problem should be. If FDC says a problem is easy but an algorithm does not find it so, this may be taken as a sign that this choice of algorithm (or algorithm parameters) is unusually bad for this problem. As an indicator of genetic algorithm performance, FDC has proved very reliable.

Are Evolutionary and Other Search Algorithms Related?

The development of the landscape model was driven by the characteristics of evolutionary search algorithms. Because it is quite general, it is worth taking the trouble to examine how search on a landscape resembles ideas of search in other domains. The division of search algorithms into navigation strategy and landscape graphs was shown above to correspond to the Artificial Intelligence community's view of search. A look at the details of heuristic search in a state space, particularly a look at earlier work [43, 142, 191], reveals that the correspondence between this form of search and search in the landscape model is rather strong.

The connection is so strong that much of what needs to be done to move between the fields can be accomplished by a simple change in the language used to describe algorithms. Substituting "state space" for "landscape," "potential solution" for "individual," "control strategy" for "navigation strategy," "heuristic function" for "fitness function," "goal" for "global optimum," and "OPEN list" for "population," much of what needs to be changed is achieved. Naturally, there are differences in these algorithms, but the similarities are very strong. Areas of difference do not so much weaken the argument of a correspondence as they indicate areas of potentially fruitful interaction between the fields.

As an example of this, consider heuristic functions and fitness functions. Both are used to label graphs in which search takes place. One big difference is the amount of research that has gone into the theory of these functions. Heuristic functions have been studied for almost 30 years [142] and there are many results regarding their properties and desirable characteristics [31]. In the world of evolutionary algorithms, fitness functions are relatively unexamined. The choice of fitness function is rarely even acknowledged as a choice. Can we transfer some of the knowledge from the theory of heuristic functions to help us with a theory of fitness functions?

The answer is Yes, and the evidence is the FDC measure. The idealized fitness function described in the previous section is not really a fitness function. A fitness function typically assigns a fixed fitness to an individual, regardless of where that individual is located. The ideal function above assigns "fitnesses" according to distance from the global optimum. But this is *exactly* what Artificial Intelligence has been explicitly seeking in heuristic functions since they were first introduced [142]. To question how well a fitness function approximates the ideal is to question how close they are to the Artificial Intelligence ideal. What we would like is not a fitness function but an heuristic function. The FDC measure arose from considering fitness functions as heuristic functions and asking what sort of a job they did. Questions about admissibility, monotonicity and informedness, from the theory of heuristic functions, can be put to fitness functions. In particular, theory of non-admissible heuristics may inform the study of fitness functions in evolutionary algorithms.

APPENDIX A

Implementation Issues for Reverse Hillclimbing

Hopefully this appendix will be helpful to anyone who implements the reverse hillclimbing algorithm. The idea is conceptually simple, but relatively efficient implementation is a little tricky.

I have implemented the reverse hillclimbing in three different ways. The first, which was the method used to produce the results in [110], used a hash table in the manner described in §4.4.2(93). Using that approach, it is necessary to keep every vertex in the basin of attraction in memory until the recursion is complete. This is due to the possibility of vertices being revisited. This algorithm was controlled using the recursive strategy described for the basic algorithm. This first implementation was sufficient to obtain basin information for steepest ascent hillclimbing on the busy beaver problem that was described in §3.8.5(60). It suffered from the need to keep the entire basin in memory (though in the cases examined in [110], the basins never contained more than 10,000 vertices), and proved far slower than the third implementation.

The second implementation tried to reduce the memory requirements of the first implementation. The experiments of Chapter 4 occasionally required identifying basins that contained over half a million vertices. Instead of keeping all the vertices of the basin of attraction in memory simultaneously and updating the statistics of vertices that were re-encountered, this version simply wrote all its discoveries to a file that was post-processed once the run completed. The resulting file contained a line of output for each vertex that was encountered during the recursion. For example, the vertex 000 in Figure 49 on page 93 would appear as the subject of three output lines, as it would be encountered during descents from 100, 010 and 001. The post-processor sorted the output file by vertex identity and then collected the statistics for the groups of lines that corresponded to each vertex. The initial phase of this implementation required very little memory, and ran very quickly. Unfortunately, it produced output files containing many tens of megabytes, which rendered this approach less useful in practice than the first.

The first two solutions both employ a strategy that resembles a depth-first search in the landscape graph. Both of the difficulties with these solutions are a result of this form of exploration. A more appealing strategy is some form of breadth-first exploration of the landscape. A breadth-first exploration based on distance from the original vertex faces the problem that a vertex v_1 at distance d_1 might not appear to be in the basin of attraction of the original vertex until the depth-first search reaches some vertex v_2 at distance $d_2 > d_1$ and discovers that vertex v_1 has some probability of ascending to v_2. Such an algorithm would be faced with storage problems similar to that of the first implementation.

The breadth-first approach can be used however if the search proceeds breadth-first according to *fitness* rather than breadth-first according to distance from the initial vertex. If all the fitter neighbors of a vertex v_1 have been completely processed by the algorithm, and have all updated their downhill neighbors that could re-ascend, then the vertex v_1 can never again be encountered. Once the downhill neighbors of the vertex v_1 are updated according to their probabilities of reaching v_1, the statistics for v_1 can be written to a file (if these are to even be recorded) and the memory allocated to v_1 can be freed. This idea has several advantages:

- Vertices in the basin will be encountered multiple times, but will only be descended from *once*. In the previous implementations, each time a vertex was reached, its basin of attraction would be calculated afresh. This provides a significant increase in speed.

- This implementation returns memory to the operating system as it does not need to keep the entire basin of attraction in memory. There is a simple condition that indicates when a vertex has been completely dealt with, and at that time its summary statistics can be gathered and the vertex discarded from memory. The peak memory load is typically 15 to 20 percent less than that required by the first implementation.

- The vertices in the basin of attraction are found in order of decreasing fitness. This allows the efficient delineation of some fraction of the upper part of the basin of attraction. The earlier implementations also provide a method of approximation, as recursive calls can be made only if the probability of ascent (or the fitness) has not fallen below some lower limit.

This method provides a large speed increase and a moderate saving in memory. Its main drawback is the memory requirement, but it has been used to find basins containing over half a million vertices.

The data structures used to implement this version are naturally more involved. The two most important operations are the fast lookup of a vertex based on its identity and a fast method for retrieving the vertex with highest fitness from those that still need to be processed. The first task is best accomplished using a hash table and the second using a priority queue (implemented as a heap). I implemented a combination hash table and priority queue to solve both problems at once. The elements of the hash table contain a pointer into the priority queue and vice-versa. This allows $O(1)$ expected vertex identity comparisons to retrieve a vertex given its identity and $O(1)$ worst-case retrieval of the unprocessed vertex with the highest fitness (the priority queue retrieves a pointer into the hash table of the highest fitness vertex in constant time, and this is dereferenced in constant time since it points directly to the stored element and does not require hashing). Insertion of a new vertex requires insertion into both the hash table and the priority queue and this requires $O(\lg n)$ comparisons (worst and average case) where n is the number of elements currently in the priority queue. Deletion requires $O(\lg n)$ comparisons (worst and average case) for similar reasons.

A somewhat simplified pseudo-code summary of this implementation is shown in Algorithm A.1. The pseudo-code assumes that the original point has already been inserted into both the hash table and the priority queue. Details of the coordination of pointers from the hash table to the priority queue and vice-versa are not shown. For more efficiency, the hash table find and insert can be done at the same time if the vertex is not present.

The above solution should not be taken as being in any way optimal. It is simply better than two others that are less sophisticated. A more efficient solution might employ a better mix of memory requirement and post-processing. If the hash table were allowed to contain only some maximum number of vertices, incomplete vertices (probably those that were judged most complete—for instance those with the highest ascent probabilities) could be written to a file for later coalescing, as in the second implementation. If a good balance could be found (and there is no reason to think this would not be straightforward, e.g., do as much in memory as you can afford), this would cap memory requirements and hopefully not demand too much disk space or post-processing time.

```
basin_size(initial_vertex)
VERTEX initial_vertex;
{
    PRIORITY_QUEUE pq;
    HASH_TABLE ht;
    VERTEX v, w;

    pq.insert(initial_vertex);
    ht.insert(initial_vertex);

    while (v = pq.get()) {
        for each $(w|v \in N_\phi(w)$ and $f(w) <_{\mathcal{F}} f(v))$ {
            if (ht.find(w)) {
                update hashed entry for w;
            }
            else {
                pq.insert(w);
                ht.insert(w);
            }
        }

        free memory associated with $v$;
    }
}
```

Algorithm A.1 A more sophisticated version of the reverse hillclimbing algorithm. This algorithm proceeds depth first (by fitness) from the original vertex. When all the vertices that could ascend to a vertex v have been considered, the memory corresponding to v can be freed.

APPENDIX B

Balanced Hillclimbing

On the six landscapes examined in Chapter 4, SA had the highest probability of locating a peak on a single hillclimb on all of them. AA was always been second. Despite this, on five of the six problems, AA is the algorithm of choice since it expects to perform fewer evaluations per uphill step than SA does, and this advantage outweighs SA's greater location probability. It was always clear that, all other things being equal, making fewer evaluations per step rather than more would result in a better algorithm. Reverse hillclimbing shows that, at least on the instances of the problems considered in Chapter 4, all other things being equal, being more exploitative rather than less will also produce a better algorithm. Since SA and AA represent two extremes when it comes to these choices in hillclimber design, it is reasonable to expect that a more balanced algorithm might perform better than either of them.

This line of reasoning leads to the idea of hillclimbers that spend some amount of time identifying uphill neighbors and then move to the highest of those found. If we restrict attention to those hillclimbers that do not restart unless they have examined all neighbors and found no improvement, these algorithms can be thought of as members of a family of hillclimbers that make use of a function called *another*. The *another* function returns a boolean value which is used to decide whether another uphill neighbor should be sought. The arguments to the function are the number of neighbors (n), the number of uphill directions located so far (u), and the number of neighbors that have been tried (t). The assumption that these hillclimbers will always continue to search for an uphill neighbor until the first is found is equivalent to saying that $another(n, 0, t) =$ TRUE for all algorithms in this family. Similarly, we can assume that the *another* function will not be called if $t = n$, i.e., if all neighbors have been tried.

Both AA and SA are members of this family (LA and MA are not since they do not, in general, move to the steepest of the uphill neighbors). In AA, the *another* function always returns FALSE, never continuing the search for uphill neighbors after the first has been located. In SA, the *another* function always returns TRUE, since SA requires the examination of all neighbors. Within this family of hillclimbers, it is easy to see the sense in which AA and SA are the extreme algorithms.

There are several obvious alternatives for the *another* function. A very simple one is a function that returns TRUE only if $u < k$ for some k (a constant or a function of n). This might be called Best-of-k hillclimbing since the resulting algorithm finds k uphill neighbors (if possible) and moves to the best of these. I will denote this algorithm by BO-k. When $k = 1$, this algorithm becomes AA and when $k = n$ it becomes SA. AA will, on average, choose the median of all the uphill neighbors for a certain amount of work. On average, BO-2 will do twice as much work but will only choose an uphill neighbor whose rank is 1/3rd amongst all uphill neighbors. A rough estimate argues that twice as much work finding uphill neighbors will not be adequately compensated for by only receiving a one-sixth increase in the ranking of the uphill neighbor chosen.

Another strategy attempts to find another uphill neighbor only if less than k neighbors have been examined for some k (a constant or a function of n). That is, the *another* function returns TRUE if $t < k$. This algorithm also degenerates to AA and SA if $k = 1$ or $k = n$. When $k = 2$ the algorithm is very similar to AA, and can be thought of as identical to AA, except it seeks a second neighbor only on those occasions when AA found an uphill neighbor on the first trial. If k is small compared to n, this algorithm can be thought of as trying for a second neighbor when AA gets lucky. This algorithm will be called Try-k hillclimbing.

Table 33. The *another* function for several hillclimbers. n is the number of neighbors, u is the number of uphill directions located so far, and t is the number of neighbors tried so far.

Hillclimber	$another(n, u, t) =$ TRUE if
AA	NEVER
SA	ALWAYS
Best-of-k	$u < k$
PA	$n - t \geq t/u$
Pess	$n - t \geq 2t/u$
Try-k	$t < k$

A more sophisticated strategy, that is close to SA in behavior, is to form an estimate of the number of neighbors that will need to be examined to locate an additional uphill neighbor. If this is greater than the remaining number of unexamined neighbors, then return FALSE. This hillclimber attempts to find many uphill neighbors, but stops looking once it appears likely that no more exist. It is similar to SA, but should use fewer evaluations, at the risk of missing uphill directions. For example, if only a single uphill direction has been found and more than half the neighbors have been examined, the *another* function of this algorithm would return FALSE. If two uphill neighbors have been found and more than two-thirds of the neighbors have been examined, it returns FALSE etc. The *another* function returns TRUE if $n-t \geq t/u$. This will be called *predictive ascent* hillclimbing, and abbreviated PA. A similar algorithm that is more pessimistic, denoted Pess, returns TRUE if $n - t \geq 2t/u$. The *another* functions for AA, SA, and the above algorithms are summarized in Table 33.

A graphical interpretation of these functions is shown in Figure 107. Any partitioning of such graphs into TRUE and FALSE regions represents a hillclimbing algorithm. These graphs also illustrate what is meant by the claim that AA and SA are in some sense algorithms at the two extremes of *another* functions.

Tables 34 to 39 show the result of hillclimbing with AA, SA, BO-2, PA, Pess, Try-2, Try-4 and Try-$n/2$ on the NK landscapes and busy beaver problems studied in Chapter 4. A † sign is used to indicate occasions when one of the six new hillclimbers has a mean number of evaluations per peak that is less than both AA and SA. The main conclusion that can be drawn from these tables is that AA is difficult to beat. Apart from one instance in which Pess beats AA (2-state busy beaver), the only hillclimbers to occasionally beat AA are the Try-n hillclimbers. There is a correspondence between the difficulty of the problem and how Try-2, Try-4 and Try-$n/2$ compare to AA. As problems get simpler, the performance of Try-n increases as n does. As problems get harder, the expected number of uphill directions can be

Table 34. The results from 10,000 hillclimbs on a 16,12 NK landscape. The table shows, for eight hillclimbers, how frequently the best 5% of 1,000 randomly located peaks were located, and the mean and standard deviation of the number of evaluations to do so.

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-$n/2$
Peaks found	653	760	738	741	674	604	639	642
Evals/peak	492	646	548	579	596	540	537	597
Evals/peak s.d.	504	620	541	593	579	514	523	604

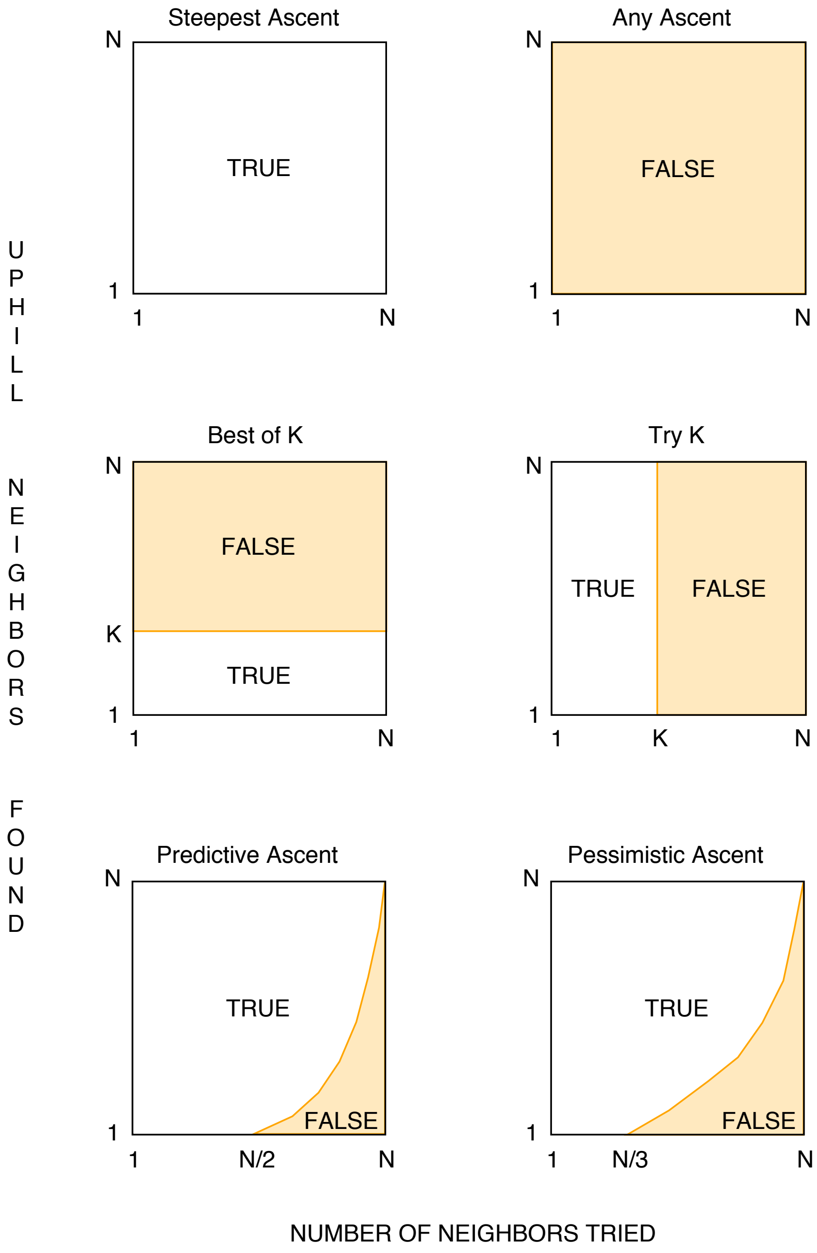


Figure 107. A graphical interpretation of six *another* functions. The X axis displays the number of neighbors examined and the Y axis the number of uphill neighbors found. The TRUE and FALSE regions indicate what the *another* function will return. All axes have a minimum of one and a maximum of n (the number of neighbors).

expected to fall and so an algorithm that spends less time looking for things that probably do not exist will tend to perform better. It should be remembered that AA is actually Try-1. Try-2 is very similar to AA—it only looks for another uphill direction if one was found on the very first neighbor examined. As the number of uphill neighbors increases, spending more time looking for them may be worthwhile. For this reason, the Try-n hillclimbers tend to do better on easy problems when n is greater.

Table 35. The results from 10,000 hillclimbs on a 16,8 NK landscape. The table shows, for eight hillclimbers, how frequently the best 5% of 700 randomly located peaks were located, and the mean and standard deviation of the number of evaluations to do so.

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-n/2
Peaks found	1661	1873	1771	1775	1693	1634	1614	1682
Evals/peak	227	332	276	303	289	235	251	280
Evals/peak s.d.	198	290	244	263	262	209	226	241

Table 36. The results from 10,000 hillclimbs on a 16,4 NK landscape. The table shows, for eight hillclimbers, how frequently the best 5% of the 180 peaks were located, and the mean and standard deviation of the number of evaluations to do so.

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-n/2
Peaks found	1600	1999	1941	1894	1798	1649	1724	1839
Evals/peak	269	397	297	358	335	268†	276	311
Evals/peak s.d.	250	346	267	327	303	236	248	273

Table 37. The results from 40 million hillclimbs on the 4-state busy beaver problem. The table shows, for eight hillclimbers, how frequently an optimal Turing machine was located, and the mean and standard deviation of the number of evaluations to do so (rounded to the nearest million).

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-$n/2$
Peaks found	44	94	68	51	59	41	44	58
Evals/peak ($\times 10^6$)	73	58	69	84	64	77	75	70
Evals/peak s.d. ($\times 10^6$)	66	56	63	95	67	66	66	76

Table 38. The results from 100,000 hillclimbs on the 3-state busy beaver problem. The table shows, for eight hillclimbers, how frequently an optimal Turing machine was located, and the mean and standard deviation of the number of evaluations to do so.

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-$n/2$
Peaks found	397	444	424	435	411	430	433	438
Evals/peak	12,670	17,732	16,607	14,527	14,013	$11,761^\dagger$	$11,903^\dagger$	13,694
Evals/peak s.d.	13,048	16,977	16,165	13,967	14,090	10,494	10,942	13,507

Table 39. The results from 100,000 hillclimbs on the 2-state busy beaver problem. The table shows, for eight hillclimbers, how frequently an optimal Turing machine was located, and the mean and standard deviation of the number of evaluations to do so.

	AA	SA	BO-2	PA	Pess	Try-2	Try-4	Try-$n/2$
Peaks found	3409	3795	3902	3991	3908	3791	3738	3874
Evals/peak	767	1003	918	786	$743^\dagger$	$718^\dagger$	$742^\dagger$	777
Evals/peak s.d.	739	939	872	745	707	671	728	748

APPENDIX C

Standard Errors from Chapter 3.

The following tables show the standard errors for all data plotted in Figures 13 to 43 in Chapter 3. Algorithm abbreviations appear in Table 1 on page 56. The various levels of achievement are listed across the top of each table. When an algorithm did not reach a performance level at least ten times, the table entry is left blank. In some cases, the levels displayed correspond only to the higher end of the performance range.

Table 40. The standard errors for CH, GA-S, GA-E and BH on the one max problem with 60 bits for the data graphed in Figure 13 on page 67.

	33	36	39	42	45	48	51	54	57	60
CH	0·16	0·32	0·53	0·85	1·34	2·08	3·38	5·63	10·4	34·0
GA-S	0·05	0·19	1·06	7·49	12·9	13·9	16·0	19·2	24·6	44·6
GA-E	0·04	0·17	0·98	6·51	10·8	12·1	14·0	16·6	20·6	35·3
BH	0·10	0·12	0·14	0·15	0·17	0·20	0·24	0·31	0·43	2·45

Table 41. The standard errors for CH, GA-S, GA-E and BH on the one max problem with 120 bits for the data graphed in Figure 14 on page 67.

	66	72	78	84	90	96	102	108	114	120
CH	0·37	0·74	1·36	2·45	4·41	7·54	12·8	22·8	45·2	1014
GA-S	0·13	1·30	14·5	20·8	24·6	29·7	38·9	58·1	901	
GA-E	0·08	0·77	8·80	12·6	14·6	18·0	23·0	32·6	82·4	
BH	0·22	0·27	0·30	0·33	0·36	0·43	0·50	0·64	0·93	7·62

Table 42. The standard errors for CH, GA-S, GA-E and BH on the fully easy problem with 10 subproblems for the data graphed in Figure 15 on page 69.

	1	2	3	4	5	6	7	8	9	10
CH	0·65	3·79	8·46	13·8	20·2	29·7	46·0	92·7	301	2137
GA-S	0·06	1·02	16·4	48·6	85·0	202	753	5099	92126	1·0E7
GA-E	0·06	1·01	16·0	42·9	68·9	118	335	1435	14537	4·3E5
BH	1·02	3·24	12·7	80·0	614	9213				

Table 43. The standard errors for CH, GA-S, GA-E and BH on the fully easy problem with 15 subproblems for the data graphed in Figure 16 on page 69.

	6	7	8	9	10	11	12	13	14	15
CH	27·5	35·4	46·2	73·8	152	401	1494	9610	1·4E5	5·0E6
GA-S	117	320	1451	11743	2·3E5	1·5E7				
GA-E	121	303	1028	5838	53218	1·0E6				
BH	373	2897	36718							

Table 44. The standard errors for CH, GA-S, GA-E and BH on the fully deceptive problem with 10 subproblems for the data graphed in Figure 17 on page 70.

	1	2	3	4	5	6	7	8	9	10
CH	0·85	3·71	15·8	47·0	101	227	774	4710	69912	3·9E6
GA-S	0·06	1·00	10·5	44·1	201	761	4164	47627	1·1E6	
GA-E	0·06	1·00	9·09	36·5	152	488	2160	14596	2·2E5	
BH	20·2	491	20146							

Table 45. The standard errors for CH, GA-S, GA-E and BH on the fully deceptive problem with 15 subproblems for the data graphed in Figure 18 on page 70.

	3	4	5	6	7	8	9	10	11	12
CH	5·10	16·5	41·8	87·0	173	384	1222	5585	46369	4·9E5
GA-S	5·63	13·5	32·8	102	264	718	2826	19927	4·3E5	
GA-E	5·59	12·6	32·5	81·5	199	488	1438	6489	47247	6·6E5
BH	6213									

Table 46. The standard errors for CH, GA-S, GA-E and BH on the distributed fully deceptive problem with 10 subproblems for the data graphed in Figure 19 on page 71.

	1	2	3	4	5	6
CH	0·30	10·0	296	7479	3·8E5	
GA-S	0·04	0·77	182	17368	3·1E6	
GA-E	0·06	1·01	194	7705	2·8E5	7·2E6
BH	21·3	510	23629			

Table 47. The standard errors for CH, GA-S, GA-E and BH on the distributed fully deceptive problem with 15 subproblems for the data graphed in Figure 20 on page 71.

	1	2	3	4	5	6	7
CH	0·15	1·92	65·2	682	6552	1·2E5	2·2E6
GA-S	0·04	0·44	20·3	1208	76983	6·1E6	
GA-E	0·04	0·44	18·1	665	10004	1·3E5	3·7E6
BH	18·0	248	4167				

Table 48. The standard errors for CH, GA-S, GA-E and BH on the busy beaver problem with 3 states for the data graphed in Figure 21 on page 72.

	0	1	2	3	4	5	6
CH	0·05	0·10	0·38	1·24	3·65	10·4	154
GA-S	0·01	0·02	0·15	1·77	13·0	48·7	787
GA-E	0·01	0·03	0·16	1·86	12·4	39·7	616
BH	0·16	0·18	0·71	1·17	1·51	12·1	126

Table 49. The standard errors for CH, GA-S, GA-E and BH on the busy beaver problem with 4 states for the data graphed in Figure 22 on page 72.

	4	5	6	7	8	9	10	11	12	13
CH	1·93	4·96	11·2	34·9	131	917	12930	1·4E5	1·3E6	5·9E6
GA-S	5·31	19·5	50·0	257	2201	43098	3·3E5	6·0E6	1·1E7	
GA-E	6·59	22·4	50·3	167	951	16545	2·1E5	2·4E6	8·1E6	
BH	1·03	1·58	4·66	16·8	78·4	1003	25372	2·4E5	1·4E6	

Table 50. The standard errors for CH, GA-S, GA-E and BH on Holland's royal road problem with $k = 4$ for the data graphed in Figure 23 on page 73.

	1	2	3	4
CH	3·23	22·2	403	1·8E5
GA-S	0·15	37·5	2070	
GA-E	0·15	34·3	1321	3·1E6
BH	2345			

Table 51. The standard errors for CH, GA-S, GA-E and BH on Holland's royal road problem with $k = 6$ for the data graphed in Figure 24 on page 73.

	1	2	3
CH	0·86	33·8	38849
GA-S	0·03	29·0	13786
GA-E	0·03	31·5	12262
BH	359	7·4E5	

Table 52. The standard errors for CH, CH-1S and CH-NJ on the one max problem with 120 bits for the data graphed in Figure 25 on page 75.

	66	72	78	84	90	96	102	108	114	120
CH	0·37	0·74	1·36	2·45	4·41	7·54	12·8	22·8	45·2	1014
CH-1S	0·31	0·80	1·81	3·52	6·19	10·4	19·4	34·5	69·1	627
CH-NJ	1·57	8·73	257							

Table 53. The standard errors for CH, CH-1S and CH-NJ on the fully easy problem with 15 subproblems for the data graphed in Figure 26 on page 75.

	6	7	8	9	10	11	12	13	14	15
CH	27·5	35·4	46·2	73·8	152	401	1494	9610	1·4E5	5·0E6
CH-1S	28·4	36·7	48·0	65·9	97·5	164	334	879	3385	35004
CH-NJ										

Table 54. The standard errors for CH, CH-1S and CH-NJ on the fully deceptive problem with 15 subproblems for the data graphed in Figure 27 on page 75.

	5	6	7	8	9	10	11	12	13	14
CH	41·8	87·0	173	384	1222	5585	46369	4·9E5		
CH-1S	42·1	86·3	161	312	708	2090	7764	41828	3·8E5	5·3E6
CH-NJ										

Table 55. The standard errors for CH, CH-1S and CH-NJ on the distributed fully deceptive problem with 15 subproblems for the data graphed in Figure 28 on page 75.

	1	2	3	4	5	6	7
CH	0·15	1·92	65·2	682	6552	1·2E5	2·2E6
CH-1S	0·11	1·47	60·5	757	7573	97328	1·8E6
CH-NJ	0·24	2·51	50·8	1068			

Table 56. The standard errors for CH, CH-1S and CH-NJ on the busy beaver problem with 4 states for the data graphed in Figure 29 on page 76.

	4	5	6	7	8	9	10	11	12	13
CH	1·93	4·96	11·2	34·9	131	917	12930	1·4E5	1·3E6	5·9E6
CH-1S	1·92	5·36	11·7	36·4	136	964	15697	1·2E5	5·6E5	1·7E6
CH-NJ	782	7718	1·0E5							

Table 57. The standard errors for CH, CH-1S and CH-NJ on Holland's royal road problem with $k = 4$ for the data graphed in Figure 30 on page 76.

	1	2	3	4
CH	3·23	22·2	403	1·8E5
CH-1S	1·93	21·1	258	14619
CH-NJ	6·35	14063		

Table 58. The standard errors for GA-S and GA-RC on the one max problem with 120 bits for the data graphed in Figure 32 on page 79.

	72	77	82	87	92	97	102	107	112	117
GA-S	1·30	11·5	20·2	23·2	26·2	30·5	38·9	53·0	108	4·6E5
GA-RC	0·91	10·2	55·0	2641	5·6E6					

Table 59. The standard errors for GA-S and GA-RC on the fully easy problem with 15 subproblems for the data graphed in Figure 33 on page 79.

	2	3	4	5	6	7	8	9	10	11
GA-S	0·43	8·11	31·8	62·8	117	320	1451	11743	2·3E5	1·5E7
GA-RC	0·42	8·56	86·0	2762	3·6E5					

Table 60. The standard errors for GA-S and GA-RC on the fully deceptive problem with 15 subproblems for the data graphed in Figure 34 on page 79.

	2	3	4	5	6	7	8	9	10	11
GA-S	0·44	5·63	13·5	32·8	102	264	718	2826	19927	4·3E5
GA-RC	0·43	6·74	38·6	616	52192					

Table 61. The standard errors for GA-S and GA-RC on the distributed fully deceptive problem with 15 subproblems for the data graphed in Figure 35 on page 79.

	1	2	3	4	5	6
GA-S	0·04	0·44	20·3	1208	76983	6·1E6
GA-RC	0·04	0·44	7·06	124	7444	8·5E5

Table 62. The standard errors for GA-S and GA-RC on the busy beaver problem with 4 states for the data graphed in Figure 36 on page 80.

	3	4	5	6	7	8	9	10	11	12
GA-S	0·68	5·31	19·5	50·0	257	2201	43098	3·3E5	6·0E6	1·1E7
GA-RC	0·87	6·87	39·0	285	4407	50396	7·1E5	7·1E6		

Table 63. The standard errors for GA-S and GA-RC on Holland's royal road problem with $k = 4$ for the data graphed in Figure 37 on page 80.

	1	2	3
GA-S	0·15	37·5	2070
GA-RC	0·16	65·7	

Table 64. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on the one max problem with 120 bits for the data graphed in Figure 38 on page 82.

	66	72	78	84	90	96	102	108	114	120
CH-1S	0·31	0·80	1·81	3·52	6·19	10·4	19·4	34·5	69·1	627
GA-E	0·08	0·77	8·80	12·6	14·6	18·0	23·0	32·6	82·4	
BH-MM	0·11	0·22	0·46	0·92	1·83	3·43	6·03	10·6	21·7	111
BH-DMM	0·12	0·21	0·37	0·66	1·15	2·05	3·59	6·29	13·3	69·5

Table 65. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on the fully easy problem with 15 subproblems for the data graphed in Figure 39 on page 82.

	6	7	8	9	10	11	12	13	14	15
CH-1S	28·4	36·7	48·0	65·9	97·5	164	334	879	3385	35004
GA-E	121	303	1028	5838	53218	1·0E6				
BH-MM	28·8	37·1	47·7	61·6	81·2	110	159	278	666	2630
BH-DMM	19320	1·7E5								

Table 66. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on the fully deceptive problem with 15 subproblems for the data graphed in Figure 40 on page 83.

	6	7	8	9	10	11	12	13	14	15
CH-1S	86·3	161	312	708	2090	7764	41828	3·8E5	5·3E6	
GA-E	81·5	199	488	1438	6489	47247	6·6E5			
BH-MM	98·8	163	257	399	727	1445	3909	13855	78320	8·2E5
BH-DMM										

Table 67. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on the distributed fully deceptive problem with 15 subproblems for the data graphed in Figure 41 on page 83.

	1	2	3	4	5	6	7
CH-1S	0·11	1·47	60·5	757	7573	97328	1·8E6
GA-E	0·04	0·44	18·1	665	10004	1·3E5	3·7E6
BH-MM	0·42	88·1	383	2028	14548	1·9E5	
BH-DMM	1·77	345	1106	5213	28631		

Table 68. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on the busy beaver problem with 4 states for the data graphed in Figure 42 on page 83.

	4	5	6	7	8	9	10	11	12	13
CH-1S	1·92	5·36	11·7	36·4	136	964	15697	1·2E5	5·6E5	1·7E6
GA-E	6·59	22·4	50·3	167	951	16545	2·1E5	2·4E6	8·1E6	
BH-MM	0·89	2·23	5·07	17·4	78·4	528	6948	1·0E5	6·3E5	2·0E6
BH-DMM	1·77	4·36	10·0	36·7	150	890	12761	1·9E5	9·2E5	

Table 69. The standard errors for CH-1S, GA-E, BH-MM and BH-DMM on Holland's royal road problem with $k = 4$ for the data graphed in Figure 43 on page 83.

	1	2	3	4	5
CH-1S	1·93	21·1	258	14619	
GA-E	0·15	34·3	1321	3·1E6	
BH-MM	0·66	11·9	93·4	911	29560
BH-DMM	6·42	452			

APPENDIX D

Function Repetition and Invariance in FDC

As noted in §5.7.1.1(139) the correlation between fitness and distance is unchanged when multiple copies of a problem are concatenated to form a larger problem. This appendix proves that this will occur for any number of copies of any function. We will denote the set of (fitness, distance) pairs of the original "base" function by $S = (X, Y)$ with cardinality n. When two copies of this function are concatenated, the (fitness, distance) pairs of the composite function will be the set $(X_i + X_j, Y_i + Y_j)$ for $1 \leq i, j \leq n$ since the fitnesses and distances of the new function are simply the sums of the fitnesses and distances in the solutions to the two subproblems. In general, let

$$S_k = (X^{(k)}, Y^{(k)}) = \bigcup_{(X_i, Y_i) \in S} (X_1 + X_2 + \cdots + X_k, Y_1 + Y_2 + \cdots + Y_k).$$

The set of (fitness, distance) pairs S_k is obtained when k copies of the base set are concatenated. Henceforth, we will abandon mention of fitness and distance, since the concepts are irrelevant to the proof. Clearly, $S_1 = (X^{(1)}, Y^{(1)}) = S = (X, Y)$. X and Y will generally be used in preference to $X^{(1)}$ and $Y^{(1)}$. The notation $X_i^{(k)}$ will be used to represent a general element of $X^{(k)}$. A simple counting argument shows that $|X^{(k)}| = |Y^{(k)}| = n^k$. The correlation $r(P)$ of any set of ordered pairs $P = (A, B)$ is given by

$$r(P) = \frac{C_{A,B}}{\sigma_A \sigma_B}$$

where

$$C_{A,B} = \frac{1}{n} \sum_i (A_i - \overline{A})(B_i - \overline{B})$$

is the covariance of A and B, $\overline{A}$ and $\overline{B}$ are the means of A and B, and

$$\sigma_A = \sqrt{\frac{\sum_{i=1}^{n} (A_i - \overline{A})^2}{n}} \quad \text{and} \quad \sigma_B = \sqrt{\frac{\sum_{i=1}^{n} (B_i - \overline{B})^2}{n}}$$

are the standard deviations of A and B.[1]

[1] The denominator in the formula for standard deviation is usually given as $n - 1$, not n. This is

Lemma 1 *For any two sets A and B both of size n,*

$$\sum(A-\overline{A})(B-\overline{B}) = \sum AB - \frac{\sum A \sum B}{n}.$$

This is a well-known statistical identity. It can be proved as follows:

$$\begin{aligned}\sum(A-\overline{A})(B-\overline{B}) &= \sum AB - \overline{B}\sum A - \overline{A}\sum B + n\overline{A}\,\overline{B}\\ &= \sum AB - n\overline{A}\,\overline{B} - n\overline{A}\,\overline{B} + n\overline{A}\,\overline{B}\\ &= \sum AB - n\overline{A}\,\overline{B}\\ &= \sum AB - \frac{\sum A \sum B}{n}.\end{aligned}$$

Lemma 2 *For all integers $k > 0$, $\sum_i X_i^{(k)} = kn^k\overline{X}$.*

The proof will rely on an identity that will be frequently used in subsequent proofs:

$$\sum_{1\le i\le n^k} X_i^{(k)} = \sum_{\substack{1\le i\le n^{k-1}\\ 1\le j\le n}} (X_i^{(k-1)} + X_j)$$

An identical result holds for $Y^{(k)}$. The identity follows from the construction of $X^{(k)}$. This double summation will hereafter be denoted $\sum\limits_{ij}$ with the understanding that the index i runs from 1 to n^{k-1} and the index j runs from 1 to n. The proof of the lemma follows:

$$\begin{aligned}\sum_i X_i^{(k)} &= \sum_{i,j}(X_i^{(k-1)} + X_j)\\ &= \sum_{i,j} X_i^{(k-1)} + \sum_{i,j} X_j\\ &= n\sum_i X_i^{(k-1)} + n^{k-1}\sum_j X_j\\ &= n\sum_i X_i^{(k-1)} + n^k\overline{X}\end{aligned}$$

because the value of the mean that is used in the formula is an *estimate* of the true mean. When the actual mean of the underlying distribution is known (as in the computation of FDC by examination of the entire representation space), n replaces $n-1$ in the computation of standard deviation [192, page 19].

Letting $T_k = \sum_i X_i^{(k)}$, we have arrived at the recurrence relation

$$T_k = \begin{cases} nT_{k-1} + n^k\overline{X} & k > 1 \\ n\overline{X} & k = 1. \end{cases}$$

Small values of k suggest that $T_k = kn^k\overline{X}$ which is quickly confirmed by induction on k.

Corollary 1 *For all integers $k > 0$, $\overline{X^{(k)}} = k\overline{X}$.*

Since $|X^{(k)}| = n^k$ and $\sum_i X_i^{(k)} = kn^k\overline{X}$,

$$\overline{X^{(k)}} = \frac{kn^k\overline{X}}{n^k} = k\overline{X}.$$

Theorem 1 *For all integers $k > 0$, $r(S_k) = r(S)$.*

This is the main result and its proof indicates that any number of copies of the base function S will exhibit the same fitness distance correlation (FDC). The proof is in two parts. In the first, it is proved that

$$C_{X^{(k)},Y^{(k)}} = kC_{X,Y} \quad \forall\ k > 0, \tag{1}$$

and in the second that

$$\sigma_{X^{(k)}}\sigma_{Y^{(k)}} = k\sigma_X\sigma_Y \quad \forall\ k > 0. \tag{2}$$

It follows immediately that

$$r(S_k) = \frac{C_{X^{(k)},Y^{(k)}}}{\sigma_{X^{(k)}}\sigma_{Y^{(k)}}} = \frac{kC_{X,Y}}{k\sigma_X\sigma_Y} = \frac{C_{X,Y}}{\sigma_X\sigma_Y} = r(S).$$

Lemma 3 *For all integers $k > 0$, $C_{X^{(k)},Y^{(k)}} = kC_{X,Y}$.*

$$\begin{aligned} C_{X^{(k)},Y^{(k)}} &= \frac{1}{n^k}\sum_i (X_i^{(k)} - \overline{X^{(k)}})(Y_i^{(k)} - \overline{Y^{(k)}}) \\ n^kC_{X^{(k)},Y^{(k)}} &= \sum_i (X_i^{(k)} - \overline{X^{(k)}})(Y_i^{(k)} - \overline{Y^{(k)}}) \\ &= \sum_i X_i^{(k)}Y_i^{(k)} - \frac{\sum_i X_i^{(k)} \sum_i Y_i^{(k)}}{n^k} \end{aligned}$$

$$
\begin{aligned}
&= \sum_i X_i^{(k)} Y_i^{(k)} - \frac{n^k k \overline{X} n^k k \overline{Y}}{n^k} \\
&= \sum_i X_i^{(k)} Y_i^{(k)} - n^k k^2 \overline{X}\,\overline{Y} \\
&= \sum_i (X_i^{(k)} Y_i^{(k)} - k^2 \overline{X}\,\overline{Y}) \\
&= \sum_{i,j} \left[(X_i^{(k-1)} + X_j)(Y_i^{(k-1)} + Y_j) - k^2 \overline{X}\,\overline{Y} \right] \\
&= \sum_{i,j} \left[X_i^{(k-1)} Y_i^{(k-1)} + Y_j X_i^{(k-1)} + X_j Y_i^{(k-1)} + X_j Y_j - k^2 \overline{X}\,\overline{Y} \right] \\
&= \sum_{i,j} \left(X_i^{(k-1)} Y_i^{(k-1)} - (k-1)^2 \overline{X}\,\overline{Y} \right) + \sum_j Y_j \sum_i X_i^{(k-1)} \\
&\quad + \sum_j X_j \sum_i Y_i^{(k-1)} + \sum_{i,j} X_j Y_j - \sum_{i,j} (2k-1) \overline{X}\,\overline{Y} \\
&= \sum_{i,j} (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) + n\overline{Y} n^{k-1} (k-1) \overline{X} \\
&\quad + n\overline{X} n^{k-1} (k-1) \overline{Y} + n^{k-1} \sum_i X_i Y_i - n^k (2k-1) \overline{X}\,\overline{Y} \\
&= \sum_{i,j} (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) + 2n^k (k-1) \overline{X}\,\overline{Y} \\
&\quad + n^{k-1} \sum_i X_i Y_i - n^k (2k-1) \overline{X}\,\overline{Y} \\
&= \sum_{i,j} (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) + n^k \overline{X}\,\overline{Y} (2(k-1) - (2k-1)) \\
&\quad + n^{k-1} \sum_i X_i Y_i \\
&= \sum_{i,j} (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) - n^k \overline{X}\,\overline{Y} + n^{k-1} \sum_i X_i Y_i \\
&= n \sum_i (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) + n^{k-1} (\sum_i X_i Y_i - n \overline{X}\,\overline{Y}) \\
&= n \sum_i (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) \\
&\quad + n^{k-1} (\sum_i X_i Y_i - \frac{\sum_i X_i \sum_i Y_i}{n}) \\
&= n \sum_i (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}}) \\
&\quad + n^{k-1} \sum_i (X_i - \overline{X})(Y_i - \overline{Y}) \qquad (3)
\end{aligned}
$$

We now examine the sum in the first term of (3), $\sum_i (X_i^{(k-1)} Y_i^{(k-1)} - \overline{X^{(k-1)}}\,\overline{Y^{(k-1)}})$.

If we let $w = k - 1$, then

$$\begin{aligned}
\sum_i (X_i^{(w)} Y_i^{(w)} - \overline{X^{(w)}}\,\overline{Y^{(w)}}) &= \sum_i X_i^{(w)} Y_i^{(w)} - \sum_i \overline{X^{(w)}}\,\overline{Y^{(w)}} \\
&= \sum_i X_i^{(w)} Y_i^{(w)} - w^2 n^w \overline{X}\,\overline{Y} \\
&= \sum_i X_i^{(w)} Y_i^{(w)} - w^2 n^w \overline{X}\,\overline{Y} - w^2 n^w \overline{X}\,\overline{Y} + w^2 n^w \overline{X}\,\overline{Y} \\
&= \sum_i X_i^{(w)} Y_i^{(w)} - w\overline{Y} w n^w \overline{X} - w\overline{X} w n^w \overline{Y} + n^w w\overline{X} w\overline{Y} \\
&= \sum_i X_i^{(w)} Y_i^{(w)} - w\overline{Y} \sum_i X_i^{(w)} - w\overline{X} \sum_i Y_i^{(w)} \\
&\quad + \sum_i \overline{X^{(w)}}\,\overline{Y^{(w)}} \\
&= \sum_i (X_i^{(w)} Y_i^{(w)} - w\overline{Y} X_i^{(w)} - w\overline{X} Y_i^{(w)} + \overline{X^{(w)}}\,\overline{Y^{(w)}}) \\
&= \sum_i (X_i^{(w)} Y_i^{(w)} - \overline{Y^{(w)}} X_i^{(w)} - \overline{X^{(w)}} Y_i^{(w)} + \overline{X^{(w)}}\,\overline{Y^{(w)}}) \\
&= \sum_i (X_i^{(w)} - \overline{X^{(w)}})(Y_i^{(w)} - \overline{Y^{(w)}}) \qquad (4)
\end{aligned}$$

Replacing w by $k - 1$ and substituting (4) back into (3), we have

$$\begin{aligned}
n^k C_{X^{(k)},Y^{(k)}} &= n \sum_i (X_i^{(k-1)} - \overline{X^{(k-1)}})(Y_i^{(k-1)} - \overline{Y^{(k-1)}}) \\
&\quad + n^{k-1} \sum_i (X_i - \overline{X})(Y_i - \overline{Y}) \\
C_{X^{(k)},Y^{(k)}} &= \frac{\sum_i (X_i^{(k-1)} - \overline{X^{(k-1)}})(Y_i^{(k-1)} - \overline{Y^{(k-1)}})}{n^{k-1}} + \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{n} \\
&= C_{X^{(k-1)},Y^{(k-1)}} + C_{X,Y} \\
&= kC_{X,Y}
\end{aligned}$$

This proves (1), the first part of the main theorem.

Lemma 4 *For all integers $k > 0$, $\sigma_{X^{(k)}} \sigma_{Y^{(k)}} = k\sigma_X \sigma_Y$*

This will be proved by examining $\sum_i (X_i^{(k)} - \overline{X^{(k)}})^2$ from which σ_X (and similarly σ_Y) are easily obtained.

$$\sum_i (X_i^{(k)} - \overline{X^{(k)}})^2 = \sum_i (X_i^{(k)} - k\overline{X})^2$$

$$
\begin{aligned}
&= \sum_{i,j} \Big[(X_i^{(k-1)} - (k-1)\overline{X}) + (X_j - \overline{X})\Big]^2 \\
&= \sum_{i,j} (X_i^{(k-1)} - (k-1)\overline{X})^2 + 2\sum_{i,j}(X_i^{(k-1)} - (k-1)\overline{X})(X_j - \overline{X}) \\
&\quad + \sum_{i,j} (X_j - \overline{X})^2 \qquad (5)
\end{aligned}
$$

Consider the middle term of (5):

$$
\begin{aligned}
2\sum_{i,j}(X_i^{(k-1)} - (k-1)\overline{X})(X_j - \overline{X}) &= 2\Big[\sum_{i,j} X_i^{(k-1)} X_j - \sum_{i,j} X_i^{(k-1)}\overline{X} \\
&\quad - \sum_{i,j}(k-1)\overline{X}X_j + \sum_{i,j}(k-1)\overline{X}^2\Big] \\
&= 2\Big[\sum_i X_i^{(k-1)} \sum_j X_j - n\overline{X}\sum_i X_i^{(k-1)} \\
&\quad - (k-1)n^{k-1}\overline{X}\sum_j X_j + (k-1)n^k\overline{X}^2\Big] \\
&= 2\Big[(k-1)n^{k-1}\overline{X}n\overline{X} - (k-1)n^k\overline{X}^2 \\
&\quad - (k-1)n^k\overline{X}^2 + (k-1)n^k\overline{X}^2\Big] \\
&= 0
\end{aligned}
$$

Therefore (5) becomes

$$
\begin{aligned}
\sum_i (X_i^{(k)} - \overline{X^{(k)}})^2 &= \sum_{i,j} (X_i^{(k-1)} - (k-1)\overline{X})^2 + \sum_{i,j} (X_j - \overline{X})^2 \\
&= n\sum_i (X_i^{(k-1)} - (k-1)\overline{X})^2 + n^{k-1}\sum_j (X_j - \overline{X})^2 \\
&= n\sum_i (X_i^{(k-1)} - \overline{X^{(k-1)}})^2 + n^{k-1}\sum_j (X_j - \overline{X})^2
\end{aligned}
$$

Letting $T_k = \sum_i (X_i^{(k)} - \overline{X^{(k)}})^2$, we have arrived at the recurrence relation

$$
T_k = \begin{cases} nT_{k-1} + n^{k-1}T_1 & k > 1 \\ \sum_i (X_i - \overline{X})^2 & k = 1. \end{cases}
$$

Small values of k suggest that $T_k = kn^{k-1}\sum_i (X_i - \overline{X})^2$ which is quickly confirmed by induction on k. Thus we have

$$\begin{aligned}
\sigma_{X^{(k)}} &= \sqrt{\frac{\sum_i \left(X_i^{(k)} - \overline{X^{(k)}}\right)^2}{n^k}} \\
&= \sqrt{\frac{kn^{k-1}\sum_i \left(X_i - \overline{X}\right)^2}{n^k}} \\
&= \sqrt{k}\sqrt{\frac{\sum_i \left(X_i - \overline{X}\right)^2}{n}} \\
&= \sqrt{k}\sigma_X
\end{aligned}$$

Similarly, $\sigma_{Y^{(k)}} = \sqrt{k}\sigma_Y$. Therefore $\sigma_{X^{(k)}}\sigma_{Y^{(k)}} = k\sigma_X\sigma_Y$ which proves (2) and completes the proof of the main theorem.

REFERENCES

[1] A. Desmond and J. Moore. *Darwin*. Penguin, London, 1992.

[2] R. Chambers. *Vestiges of the Natural History of Creation*. Chicago University Press, Chicago IL, 1994.

[3] J. Huxley. *Evolution, The Modern Synthesis*. Allen and Unwin, London, 1942.

[4] E. Mayr. *The Growth of Biological Thought*. Harvard University Press, Cambridge, MA, 1982.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[6] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, Jan 1994. Special Issue on Evolutionary Computation.

[7] C. G. Langton. Artificial life. In C. G. Langton, editor, *Artificial Life*, volume 1, pages 1–47, Santa Fe, NM, 1989. Santa Fe Institute Studies in the Sciences of Complexity., Addison-Wesley.

[8] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, New York, 1966.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[10] I. Rechenberg. *Evolutionsstrategie: Optimierung Techniquer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.

[11] H.-P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. *Interdisciplinary systems research*, 26, 1977. Birkhäuser, Basel.

[12] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–24, 1993.

[13] D. E. Goldberg, K. Deb, and B. Korb. Don't worry, be messy. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 24–30, San Mateo, CA, 1991. Morgan Kaufmann.

[14] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[15] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[16] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the sixth international congress of genetics*, volume 1, pages 356–366, 1932.

[17] J. B. S. Haldane. A mathematical theory of natural and artificial selection, part viii: Metastable populations. *Transactions of the Cambridge Philosophical Society*, 27:137–142, 1931.

[18] N. Eldredge. *Macroevolutionary Dynamics: Species, Niches and Adaptive Peaks.* McGraw-Hill, 1989.

[19] G. G. Simpson. *Tempo and Mode in Evolution.* Columbia University Press, New York, 1944.

[20] S. Wright. The distribution of gene frequencies in populations. In *Proceedings of the National Academy of Science*, volume 23, pages 307–320, 1937.

[21] R. Lande. Natural selection and random genetic drift in phenotype evolution. *Evolution*, 30:314–334, 1976.

[22] D. Sherrington and S. Kirkpatrick. Solvable model of a spin glass. *Physical Review Letters*, 32:1792–1796, 1975.

[23] C. Amitrano, L. Peliti, and M. Saber. A spin-glass model of evolution. In A. Perelson and S. A. Kauffman, editors, *Molecular Evolution on Rugged Landscapes: Proteins, RNA and the Immune System*, pages 27–38, Redwood City, CA, Dec 1987. Santa Fe Institute Studies in the Sciences of Complexity, volume IX, Addison-Wesley.

[24] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[25] M. Minsky. Steps toward artificial intelligence. In E. A. Feigenbaum and J. Friedman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, 1963. This originally appeared in *Proc, IRE*, vol. 49, pp. 8–30, 1961.

[26] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, MA, 1987.

[27] G. J. E. Rawlins. Introduction. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 1–10. Morgan Kaufmann, San Mateo, CA, 1991.

[28] J. C. Culberson and G. J. E. Rawlins. Genetic algorithms as function optimizers. Unpublished manuscript., December 1992.

[29] J. C. Culberson. Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, 2(3):279–311, 1995.

[30] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975. Dissertation Abstracts International 36(10), 5410B. (University Microfilms No. 76–9381).

[31] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[32] S. Amarel. On representations of problems of reasoning about actions. In D. Michie, editor, *Machine Intelligence*, volume 3, pages 131–171. Edinburgh University Press, 1968.

[33] R. E. Korf. Macro-Operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[34] T. Shiple, P. Kollaritsch, D. J. Smith, and J. Allen. Area evaluation metrics for transistor placement. In *Proceedings of the IEEE International Conference on Computer Design: VLSI Computers and Processors*, pages 428–433, Washington, DC, October 1988. Computer Society of the IEEE.

[35] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[36] 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, *Proceedings of the First European Conference on Artificial Life. Toward a Practice of Autonomous Systems*, pages 245–254, Cambridge, MA, 11–13 Dec 1992. MIT Press.

[37] S. Wright. Surfaces of selective value revisited. *American Naturalist*, 131(1):115–123, 1988.

[38] T. C. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995. (to appear).

[39] T. C. Jones and S. Forrest. Genetic algorithms and heuristic search. Technical Report 95–02–021, Santa Fe Institute, Santa Fe, NM, February 1995. Available via ftp from ftp.santafe.edu in pub/terry/gahs.ps.gz.

[40] D. E. Knuth. *The Art of Computer Programming*, volume 2 : Sorting and Searching. Addison-Wesley, Reading, MA, 2nd edition, 1980.

[41] P. Kanerva. *Sparse Distributed Memory.* Bradford Books, MIT Press, Cambridge MA, 1988.

[42] F. Harary, R. Z. Norman, and D. Cartwright. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley & Sons, New York, 1965.

[43] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, 1971.

[44] E. Rich. *Artificial Intelligence.* McGraw-Hill, New York, 1983.

[45] R. B. Banerji. *Theory of problem Solving: An Approach to Artificial Intelligence.* Modern Analytic and Computational Methods in Science and Mathematics. American Elsevier, New York, NY, 1969.

[46] D. J. Smith, February 1995. Personal communication.

[47] G. Liepins and M. D. Vose. Representational issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:101–115, 1990.

[48] K. Mathias and L. D. Whitley. Remapping hyperspace during genetic search: Canonical delta folding. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 167–186, San Mateo, CA, 1993. Morgan Kaufmann.

[49] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9(1):9–21, June 1992.

[50] C. G. Shaefer. The ARGOT strategy: Adaptive representation genetic optimizer technique. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–55, Hillsdale NJ, 1987. Lawrence Erlbaum Associates.

[51] L. D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84, San Mateo, CA, 1991. Morgan Kaufmann.

[52] A. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88, 1992.

[53] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[54] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[55] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[56] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1991. Morgan Kaufmann.

[57] K. E. Kinnear Jr., editor. *Advances in Genetic Programming*, Cambridge, MA, 1994. MIT Press.

[58] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230, Hillsdale, NJ, 1987. Lawrence Erlbaum.

[59] L. D. Whitley, T. Starkweather, and F. D'Ann. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[60] B. Manderick, M. De Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150, San Mateo, CA, 1991. Morgan Kaufmann.

[61] K. Mathias and L. D. Whitley. Genetic operators, the fitness landscape and the traveling salesman problem. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature*, volume 2, pages 219–228, Amsterdam, The Netherlands, 1992. Elsevier Science Publishers B.V.

[62] P. F. Stadler and W. Schnabl. The landscape of the traveling salesman problem. *Physics Letters A*, 161:337–344, 1992.

[63] E. D. Weinberger. Fourier and taylor series on fitness landscapes. *Biological Cybernetics*, 65:321–330, 1990.

[64] E. D. Weinberger. Measuring correlations in energy landscapes and why it matters. In H. Atmanspacher and H. Scheingraber, editors, *Information Dynamics*, pages 185–193. Plenum Press, New York, 1991.

[65] E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[66] R. Parsons, S. Forrest, and C. Burks. Genetic operators for the DNA fragment assembly problem. *Machine Learning*, 1995. (in press).

[67] N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann.

[68] D. E. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 24–26 July 1985.

[69] T. Starkweather, S. McDaniel, K. Mathias, and L. D. Whitley. A comparison of genetic sequencing operators. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69–76, San Mateo, CA, 1991. Morgan Kaufmann.

[70] J. Dzubera and L. D. Whitley. Advanced correlation analysis of operators for the traveling salesman problem. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving From Nature – PPSN III, volume 866 of Lecture Notes in Computer Science*, pages 68–77, Berlin, 1994. Springer-Verlag.

[71] M. Gorges-Schleuter. ASPARAGOS An asynchronous parallel genetic optimization strategy. In J. D. Schaffer, editor, *Proceedings of the Third International*

Conference on Genetic Algorithms, pages 422–427, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[72] H. Mühlenbein. Evolution in time and space – The parallel genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 316–337, San Mateo, CA, 1991. Morgan Kaufmann.

[73] P. Moscato and M. G. Norman. A "memetic" approach for the travelling salesman problem—implementation of a computational ecology for combinatorial optimisation on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*, Amsterdam, 1992. IOS Press.

[74] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 2nd edition, 1994.

[75] K. A. De Jong. Genetic algorithms; A 10 year perspective. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 169–177, Hillsdale, NJ, 24–26 July 1985. Carnegie Mellon University, Lawrence Erlbaum.

[76] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.

[77] J. J. Hopfield and D. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[78] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidlines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[79] W. Siedlecki and J. Sklansky. Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 141–150, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[80] L. Davis and M. Steenstrup. Genetic algorithms and simulated annealing: An overview. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 1–11. Pitman (Morgan Kaufmann), London, 1987.

[81] D. Orvosh and L. Davis. Shall we repair? Genetic algorithms, combinatorial optimization and feasibility constraints. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 650, San Mateo, CA, 1993. Morgan Kaufmann.

[82] W. Banzhaf. Genetic programming for pedestrians. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 628, San Mateo, CA, 1993. Morgan Kaufmann.

[83] R. Nakano and T. Yamada. Conventional genetic algorithm for job shop problems. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 474–479, San Mateo, CA, 1991. Morgan Kaufmann.

[84] J. C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.

[85] D. L. Battle and M. D. Vose. Isomorphisms of genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 242–251, San Mateo, CA, 1991. Morgan Kaufmann.

[86] J. Horn, D. E. Goldberg, and K. Deb. Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving From Nature – PPSN III, volume 866 of Lecture Notes in Computer Science*, pages 149–158, Berlin, 1994. Springer-Verlag.

[87] W. B. Provine. *Sewall Wright and Evolutionary Biology.* University of Chicago Press, Chicago, IL, 1986.

[88] D. B. Fogel and J. W. Atmar. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63:111–114, 1990.

[89] L. J. Eshelman and J. D. Schaffer. Crossover's niche. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (ICGA 1993)*, pages 9–14, San Mateo, CA, 1993. Morgan Kaufmann.

[90] W. M. Spears. Crossover or mutation? In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 221–237, San Mateo, CA, 1993. Morgan Kaufmann.

[91] D. B. Fogel. Applying evolutionary programming to selected control problems. *Computers Math. Applic.*, 27(11):89–104, 1994.

[92] D. B. Fogel and L. C. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, 32:171–182, 1994.

[93] U.-M. O'Reilly and F. Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving From Nature – PPSN III, volume 866 of Lecture Notes in Computer Science*, pages 397–406, Berlin, 1994. Springer-Verlag.

[94] U.-M. O'Reilly and F. Oppacher. Hybridized crossover-based search techniques for program discovery. Technical Report 95–02–007, Santa Fe Institute, Santa Fe, NM, February 1995.

[95] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[96] S. Forrest and M. Mitchell. Towards a stronger building-blocks hypothesis: Effects of relative building-block fitness on GA performance. In *Foundations of Genetic Algorithms*, volume 2, pages 109–126, Vail, Colorado, 1993. Morgan Kaufmann.

[97] E. Brindle. *Genetic Algorithms for Function Optimization.* PhD thesis, University of Alberta, Edmonton, Canada., 1981. Available as TR 81–2, Department of Computer Science, University of Alberta.

[98] L. D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, CA, June 4–7 1989. Morgan Kaufmann.

[99] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 69–93, San Mateo, CA, 1991. Morgan Kaufmann.

[100] K. A. De Jong. Genetic algorithms are NOT function optimizers. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 5–17, San Mateo, CA, 1993. Morgan Kaufmann.

[101] L. D. Whitley. Introduction. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 1–4, San Mateo, CA, 1993. Morgan Kaufmann.

[102] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, 2nd edition, 1992.

[103] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, Jan/Feb 1986.

[104] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 6, pages 74–88. Pitman (Morgan Kaufmann), London, 1987.

[105] K. Deb and D. E. Goldberg. Sufficient conditions for deceptive and easy binary functions. Technical report, University of Illinois, Urbana-Champaign, 1992. IlliGAL Report No 92001. Available via ftp from gal4.ge.uiuc.edu in pub/papers/IlliGALs/92001.ps.Z.

[106] T. Rado. On non-computable functions. *Bell System Technical Journal*, 41:877–884, 1962.

[107] A. H. Brady. The conjectured highest scoring machines for Rado's $\sigma(k)$ for the value $k = 4$. *IEEE Transactions on Electronic Computers*, EC-15:802, 1966.

[108] A. H. Brady. The determination of the value of Rado's noncomputable function $\sigma(k)$ for four-state Turing machines. *Mathematics of Computation*, 40(162):647–665, April 1983.

[109] H. Marxen and J. Buntrock. Attacking the busy beaver 5. *Bulletin of The European Association for Theoretical Computer Science*, 40:247–251, 1990.

[110] T. C. Jones and G. J. E. Rawlins. Reverse hillclimbing, genetic algorithms and the busy beaver problem. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (ICGA 1993)*, pages 70–75, San Mateo, CA, 1993. Morgan Kaufmann.

[111] R. G. Palmer, July 1992. Personal communication.

[112] J. H. Holland. Royal road functions. Internet Genetic Algorithms Digest v7n22. Available via ftp from ftp.santafe.edu in pub/terry/jhrr.tar.gz, Aug 12 1993.

[113] J. R. Levenick. Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 123–127, San Mateo, CA, 1991. Morgan Kaufmann.

[114] R. G. Palmer and C. M. Pond. Internal field distributions in model spin glasses. *Journal of Physics F*, 9(7):1451–1459, 1979.

[115] K. A. De Jong, W. M. Spears, and D. F. Gordon. Using Markov chains to analyze GAFOs. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann. (To appear).

[116] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113, September 1994.

[117] S. A. Kauffman. Adaptation on rugged fitness landscapes. In D. Stein, editor, *Lectures in the Sciences of Complexity*, volume 1, pages 527–618. Addison-Wesley Longman, 1989.

[118] S. A. Kauffman. *The Origins of Order; Self-Organization and Selection in Evolution.* Oxford University Press, New York, 1993.

[119] W. A. Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs.* PhD thesis, University of Southern California, Los Angeles, CA, April 1994.

[120] W. A. Tackett. Greedy recombination and genetic search on the space of computer programs. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann. (To appear).

[121] N. J. Nilsson and D. Rumelhart. Approaches to Artificial Intelligence. Technical Report 93–08–052, Santa Fe Institute, Santa Fe, NM, 1993. Summary of workshop held November 6–9, 1992. Available via ftp from ftp.santafe.edu in pub/Users/mm/approaches/approaches.ps.

[122] J.-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.

[123] S. Loyd. *Mathematical Puzzles of Sam Loyd.* Dover, new York, 1959.

[124] J. Slagle. A computer program for solving problems in freshman calculus (SAINT). *Journal of the Association for Computing Machinery*, 10(4):507–520, 1963. This also appears in E. A. Feigenbaum and J. Friedman (Eds.) *Computers and Thought* pp. 191–203. McGraw-Hill, New York, 1963.

[125] D. Partridge. *A New Guide to Artificial Intelligence.* Ablex Publishing Co., Norwood, NJ, 1991.

[126] J. L. Noyes. *Artificial Intelligence with Common Lisp.* D. C. Heath, Lexington, MA, 1992.

[127] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, Reading, MA, 3 edition, 1992.

[128] G. F. Luger and A. Stubblefield. *Artificial Intelligence and the Design of Expert Systems.* Benjamin/Cummins, Redwood City CA, 2nd edition, 1993.

[129] W. B. Gevarter. *Artificial Intelligence, Expert Systems, Computer Vision and natural Language Processing.* Noyes Publications, Park Ridge, NJ, 1984.

[130] S. L. Tanimoto. *The Elements of Artificial Intelligence.* W. H. Freeman and Co., New York, 1990.

[131] D. W. Patterson. *Introduction to Artificial Intelligence and Expert Systems.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[132] M. Ginsberg. *Essentials of Artificial Intelligence.* Morgan Kaufmann, San Mateo, CA, 1993.

[133] A. J. Gonzalez and D. D. Dankel. *The Engineering of Knowledge-Based Systems.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[134] S. J. Scown. *The Artificial Intelligence Experience: An Introduction.* Digital Equipment Corporation, 1985.

[135] R. J. Schalkoff. *Artificial Intelligence: An Engineering Approach.* McGraw-Hill, New York, 1990.

[136] G. W. Ernst and A. Newell. *GPS:A case Study in Generality and Problem Solving.* Academic Press, New York, 1969.

[137] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, CA, 1980.

[138] I. Pohl. Bi-Directional search. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 127–140, New York, 1971. American Elsevier.

[139] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[140] P. Helman. An algebra for search problems and their solutions. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 28–90. Springer Verlag, New York, 1988.

[141] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.

[142] J. Doran and D. Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society of London (A)*, 294:235–259, 1966.

[143] J. Horn and D. E. Goldberg. Genetic algorithm difficulty and the modality of fitness landscapes. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann. (To appear).

[144] A. D. Bethke. *Genetic Algorithms as Function Optimizers.* PhD thesis, University of Michigan, Ann Arbor, MI, 1981.

[145] L. D. Whitley. Fundamental principles of deception in genetic search. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 221–241, San Mateo, CA, 1991. Morgan Kaufmann.

[146] R. Das and L. D. Whitley. The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 166–173, San Mateo, CA, 1991. Morgan Kaufmann.

[147] J. J. Grefenstette. Deception considered harmful. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 75–91, San Mateo, CA, 1993. Morgan Kaufmann.

[148] J. D. Schaffer, L. J. Eshelman, and D. Offutt. Spurious correlations and premature convergence in genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 102–112, San Mateo, CA, 1991. Morgan Kaufmann.

[149] H. Kargupta and D. E. Goldberg. Decision making in genetic algorithms: A signal-to-noise perspective. Technical Report #94004, IlliGAL, University of Illinois, 1994.

[150] H. Kargupta. Signal-to-noise, crosstalk, and long range problem difficulty in genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, CA, 1995. Morgan Kaufmann.

[151] Y. Davidor. Epistasis variance: A viewpoint on GA-hardness. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 23–35, San Mateo, CA, 1991. Morgan Kaufmann.

[152] R. Tanese. *Distributed Genetic Algorithms for Function Optimization*. PhD thesis, University of Michigan, Ann Arbor, MI, 1989.

[153] D. Lane, November 1994. Personal communication.

[154] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 4:415–444, 1989.

[155] T. C. Jones. A description of Holland's royal road function. *Evolutionary Computation*, 2(4):411–417, 1995.

[156] G. Liepins and M. D. Vose. Deceptiveness and genetic algorithm dynamics. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 36–50, San Mateo, CA, 1991. Morgan Kaufmann.

[157] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. *Machine Learning*, 13:285–319, 1993.

[158] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 93–108, San Mateo, CA, 1993. Morgan Kaufmann.

[159] R. A. Caruana and J. D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Fifth International Conference on Machine Learning*, pages 153–161, Los Altos, CA, June 12–14 1988. Morgan Kaufmann.

[160] L. Davis. Bit climbing, representational bias and test suite design. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 18–23, San Mateo, CA, 1991. Morgan Kaufmann.

[161] S. Wright. Evolution in Mendelian populations. *Genetics*, 16:97–159, 1931.

[162] W. Fontana, P. F. Stadler, E. G. Bornberg-Bauer, T. Griesmacher, I. Hofacker, M. Tacker, P. Tarazona, E. D. Weinberger, and P. Schuster. RNA folding and combinatory landscapes. *Physical Review E*, 47(3):2083–2099, 1993.

[163] M. Huynen. *Evolutionary Dynamics and Pattern Generation in the Sequence and Secondary Structure of RNA: A Bioinformatic Approach.* PhD thesis, University of Utrecht, Netherlands, September 1993.

[164] E. D. Weinberger. Local properties of Kauffman's N-k model: A tunably rugged energy landscape. *Physical Review A*, 44(10):6399–6413, November 1991.

[165] K. E. Kinnear Jr. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computing*, pages 142–47, 1994.

[166] P. F. Stadler and W. Grüner. Anisotropy in fitness landscapes. *Journal of Theoretical Biology*, 165:373–388, 1993.

[167] P. Schuster and P. F. Stadler. Landscapes: Complex optimization problems and biopolymer structures. *Computers Chem.*, 18:295–314, 1994.

[168] P. F. Stadler. Linear operators on correlated landscapes. *J. Physique*, 4:681–696, 1994.

[169] T. E.. Davis and J. C. Principe. A simulated annealing like convergence theory for the simple genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 174–181, San Mateo, CA, 1991. Morgan Kaufmann.

[170] M. D. Vose. Modeling simple genetic algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 63–73, San Mateo, CA, 1993. Morgan Kaufmann.

[171] J. Suzuki. A Markov chain analysis on a genetic algorithm. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (ICGA 1993)*, pages 146–153, San Mateo, CA, 1993. Morgan Kaufmann.

[172] G. Rudolph. Convergence analysis of conventional genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, Jan 1994. Special Issue on Evolutionary Computation.

[173] D. E. Goldberg and P. Segrest. Finite Markov chain analysis of genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 1–8. Lawrence Erlbaum, Hillsdale, NJ, 1987.

[174] S. W. Mahfoud. Finite Markov chain models of an alternative selection strategy for the genetic algorithm. *Complex Systems*, 7(2):155–170, April 1993.

[175] J. Horn. Finite Markov chain analysis of genetic algorithms with niching. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (ICGA 1993)*, pages 110–117, San Mateo, CA, 1993. Morgan Kaufmann.

[176] L. D. Whitley and N.-W. Yoo. Modeling simple genetic algorithms for permutation problems. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann. (To appear).

[177] G. Wagner, January 1995. Personal communication.

[178] L. D. Whitley, July 1994. Personal communication.

[179] M. D. Vose, July 1994. Personal communication.

[180] L. Altenberg. The schema theorem and Price's theorem. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms*, volume 3, San Mateo, CA, 1995. Morgan Kaufmann. (To appear).

[181] B. T. Lowerre and R. D. Reddy. The Harpy speech understanding system. In W. A. Lea, editor, *Trends in Speech Recognition.* Prentice Hall, Englewood Cliffs, NJ, 1980.

[182] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 31:498–516, 1973.

[183] K. D. Boese. Cost versus distance in the traveling salesman problem. Unpublished preliminary report., 1995.

[184] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.

[185] L. Hagen and A. B. Kahng. Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning. *IEEE Transactions on Computer Aided Design*, 1995. (to appear).

[186] W. Ruml, J. T. Ngo, J. Marks, and S. Shieber. Easily searched encodings for number partitioning. *Journal of Optimization Theory and Applications*, 1995. (to appear).

[187] N. Karmarkar and R. M. Karp. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, University of California, Berkeley, Berkeley, CA, 1982.

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

[189] M. R. Garey and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[190] P. Orponen, K.-I. Ko, U. Schöning, and O. Watanabe. Instance complexity. *Journal of the Association for Computing Machinery*, 41(1):96–121, January 1994.

[191] J. Doran. New developments of the graph traverser. In E. Dale and D. Michie, editors, *Machine Intelligence*, volume 2, pages 119–135, New York, 1967. American Elsevier.

[192] N. T. J. Bailey. *Statistical Methods in Biology*. Cambridge University Press, Cambridge, UK, 3rd edition, 1995.