A CONNECTIONIST MACHINE FOR GENETIC HILLCLIMBING

by

David H. Ackley
Carnegie Mellon University

KLUWER ACADEMIC PUBLISHERS
Boston/Dordrecht/Lancaster

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061, USA

Distributors for the UK and Ireland:
Kluwer Academic Publishers
MTP Press Limited
Falcon House, Queen Square
Lancaster LA1 1RN, UNITED KINGDOM

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

Ackley, David H.
A connectionist machine for genetic hillclimbing.

(The Kluwer international series in engineering and computer science ; SECS 28)
Originally presented as the author's thesis (Ph. D.)—Carnegie Mellon University, Pittsburgh, 1987.
Bibliography: p.
Includes index.
1. Artificial intelligence—Data processing.
I. Title. II. Series.
Q336.A25 1987 006.3 87-13536
ISBN 0-89838-236-X

Printed in the United States of America

Chapter 1
Introduction

This dissertation describes, demonstrates, and analyzes a new search technique called *stochastic iterated genetic hillclimbing* (SIGH). SIGH performs function optimizations in high-dimensional, binary vector spaces. Although the technique can be characterized in abstract computational terms, it emerged from research into massively parallel connectionist learning machines, and it has a compact implementation in a connectionist architecture. The behavior generated by the machine displays elements of two existing search techniques—stochastic hillclimbing and genetic search.

Viewed in the context of search, the central claim of the dissertation is this:

> *Genetic hillclimbing algorithms*—combinations of genetic search and hillclimbing—can offer dramatic performance improvements over either technique alone.

SIGH is the principal genetic hillclimbing algorithm considered in the dissertation, but a second algorithm, called simply *iterated genetic hillclimbing* (IGH), is defined and explored as well. Compared to SIGH, IGH is more naturally suited to sequential machines, and empirical tests show that it can far outperform both pure genetic search algorithms and pure hillclimbing algorithms on difficult problems.

This chapter describes the class of problems that the model is designed to solve, and then provides an overview of the remainder of the thesis.

1.1 Satisfying hidden strong constraints

The way in which a problem is stated can make all the difference in the world

when it comes to deciding how to go about trying to solve it. A great variety of *problem formulations* have been employed in the artificial intelligence and computer science literatures. Speaking very generally, a piece of computational research can be viewed as having two basic parts—a problem to be solved, and a computational method for solving the problem. In many cases, particular problems have been adopted by groups of researchers for a period of time, and the motivation for the problem formulation can be assumed to be familiar to the reader. In other cases, the adopted problem is a new twist on an old problem, or a new combination of old problems, or occasionally something completely different, and in such cases more effort must be spent on motivation. Sometimes an apparently small change in a problem formulation can have widespread unforeseen effects on the method of solution.

The problem formulation employed in this dissertation can be viewed as a form of *constraint satisfaction search*, an approach that has a well-established heritage in artificial intelligence research. However, there are many ways in which this general characterization can be made specific. In particular, a distinction can be drawn between satisfying *strong* constraints and satisfying *weak* constraints . By definition, all strong constraints must be satisfied for the problem to be considered solved. If, for example, one poses the problem of finding a checkmate in terms of attacking the opponent's king and eliminating all escapes, a computational method that searched through many alternatives and ended up allowing the king *just one move* could not be considered to have solved the problem at all.

By contrast, in some kinds of problems the constraints are not so all-or-none; often it is the case that it is simply impossible to satisfy all the constraints. For example, in the vision problem of inferring a three-dimensional scene from a two-dimensional image, it is very useful to include the weak constraint that nearby points in the image should normally be assigned nearby depths in three-space, capturing a simple notion of smooth surfaces. Of course, there will usually be occluding edges in the scene, and an optimal solution would simply go ahead and violate the smoothness constraint at such points. Weak constraints specify *desirable* properties of solutions; strong constraints specify *mandatory* properties of solutions.

A significant body of artificial intelligence research, and much theoretical computer science algorithms research, has effectively worked within a strong constraint satisfaction context. In one sense, the whole point of an *algorithm* for solving a problem is that it is provides a strong guarantee about some properties that computed solution will possess. A sorting algorithm, for example, *guarantees* that the final arrangement of the data will be properly ordered. Nobody has ever seriously proposed (to my knowledge, at least) a strategy that only promises to get the data "mostly" sorted.[1]

[1] More recently, some computer science theorists have taken steps away from the strong constraint position, investigating approximate or probabilistic algorithms that have much more of the weak constraint character. Increased understanding of the computational

numbers, find a point in the domain that produces a value greater than or equal to some specified criterion. If the specified criterion is the maximum possible function value, then only globally maximal points count as solutions; if the criterion is lower, the solution set may include points with a range of function values. Alternatively, one may be interested in minimizing the function values rather maximizing—in this dissertation both descriptions will be used as seems appropriate, since different metaphors are called to mind by the choice of direction, e.g., hillclimbing versus gradient descent. Beyond the choice of which extremal value is sought, there are a tremendous number of more substantial variations that have been studied, adding assumptions about the domain, the range, and/or the nature of the functions-to-be-optimized.

In some applications of function optimization, it is assumed that the problem solver has a fixed amount of time in which to search, and it must find the best point it can in that amount of time. Such formulations use function optimization in a weak constraint context—in principle, *every* point in the space is a solution, but some solutions are better than others. By contrast, with the strong constraint approach employed in this dissertation, there is a strict *success criterion* that divides the space into a set of solution points and a set of non-solution points. Finding any solution point suffices to end the computation, and until a problem solver does find a solution point, the computation remains unfinished.

It may seem that the distinction between strong and weak constraints amounts to the distinction between finding the *best* point in the space and finding a *good* point, but that viewpoint can be misleading here. The key issue is whether the *definition of a solution* is strict or not. For example, suppose the maximum value of some function was 100, but that any point scoring above 50 serves the purposes of the computation equally well, and every point scoring below 50 is unacceptable. That would be a strong constraint situation—the first score of 50 or more ends the computation, and no below-50 score counts as a solution. Although in the simulations presented in later chapters, the search strategies *are* required to find the maximum achievable score, such success criteria were chosen so that the problems-to-be-solved would be clear, and to generate problems hard enough that pure hillclimbing strategies would run into difficulty—it often happens that a simple hillclimber can very quickly find a point scoring within, say, 10% of the maximum.

The distinction between strong and weak constraints is similar to, but separable from, Simon's distinction between "optimizing" and "satisficing" (Simon, 1981). The two views can be connected if Simon's terms are understood as referring not to possible function values but to possible stopping criteria. "Optimal" stopping criteria do not allow termination until all constraints are met (whether there are few or many possible solutions), whereas "satisficing" stopping criteria will in principle accept any point that a search strategy ends up offering. In this dissertation, I speak in terms of "strong" and "weak" constraints to avoid overloading the term "optimizing."

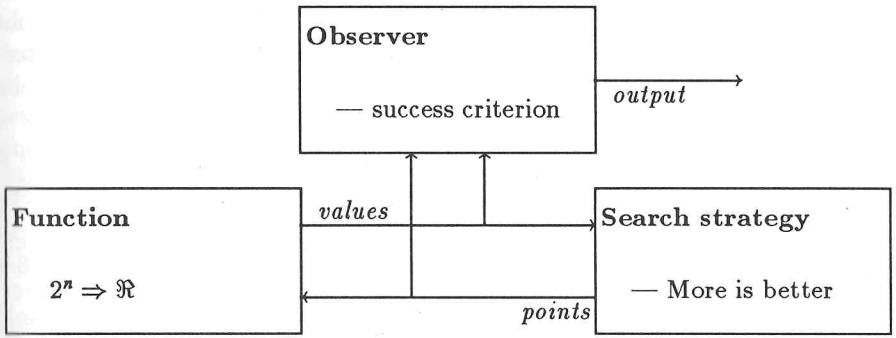


Figure 1–1. A block diagram of the problem formulation used in this dissertation.

In this dissertation, the function-to-be-optimized is assumed to be a "black box"—in other words, no information about the structure and internal workings of the function is available *a priori* to the problem solver. The only way to get information about the function is to supply inputs to the black box and observe resulting function values. This is a very important point, because it implies that search is required to solve the problem. Strategies that exploit advance knowledge of the structure of the function are ruled out. Furthermore, the problem solver is not even informed what the constraints are—in particular, neither the maximum achievable value nor the success criterion is available to the optimizer. The only information provided is that higher function values indicate points that satisfy the constraints better than points with lower function values. A consequence of this is that if some point scoring x does not satisfy all the constraints and thereby end the computation, then no point scoring y such that $y \leq x$ will satisfy all the constraints either. Another consequence is: *The search strategy has no way of knowing when the problem is solved!*

Figure 1–1 shows the structure of the problem formulation used in this dissertation. It has some unusual (and perhaps idiosyncratic) characteristics. To begin with, there are no downward arrows—the observer decides what counts as a solution, and presumably moves on to other problems once a solution has been discovered, without affecting the cross-coupled search/function system in any way. In effect, the search strategy is playing the role of the generator in an overall generate-and-test environment. The tester evaluates the function at the points specified by the generator, and compares the function value to the required threshold value. If the threshold is met or exceeded, the computation terminates, supplying the just-evaluated point as the output. If not, the function value is returned to the generator as feedback, and the computation continues.

The generate-and-test paradigm is familiar as one of the "weak methods" (Newell & Simon, 1972) studied in the artificial intelligence literature, and so

is the difficulty that the paradigm is likely to engender—the "combinatoric explosion." As a search space grows linearly in the number of dimensions, the number of possible states grows exponentially, and—if the generator is "stupid"—so does the time required to find a solution. This standard problem, in turn, has a standard solution: Put more knowledge into the generator.

Now, one way this could be done is by *programming* knowledge about the function space into the generator, so that it would search mostly in promising regions of the function space, and many lines of research have taken that tack. In the present context, such a move cannot be made without violating the black box assumption. The motivation for this particular division of labor is to create a situation that prominently emphasizes both *learning while searching* and *sustained exploration*. To solve the problem reasonably quickly, the generator should adapt its behavior based on the function values it receives as feedback, but at the same time it should not allow the feedback to trick it into permanently searching in a limited part of the space.

The black box assumption has another important implication in terms of function optimization strategies. Since the space of possible inputs is defined by the cross product of the possible values on each input dimension, the black box approach is really only aimed at optimizing *complete functions*, which provide function values for every point in the input space. However, sometimes we are interested in optimizing *partial functions*, which are undefined for some subset of all possible inputs to the function. The definition of "legal inputs" is dependent on the particular function at hand, and therefore is "in the box" with the function. It is not directly accessible to the optimizer. If we are employing a partial function, what should be done when the optimizer requests the evaluation of a point for which the function is undefined? The solution adopted in this dissertation is to extend the definition of partial functions to make them complete functions by adding "penalty terms" that score undefined points based on how far they are from the nearest defined point. The basic idea is to make undefined points score worse than defined points, so that the optimizer will head for defined regions of the space under its own power.[2] It is then the modified, complete function that is actually put into the box for optimization.

So, assuming we are now dealing only with complete functions, a decision must be made concerning how an input to the function is to be represented. In this dissertation, an input is represented as a bit vector of length n. The function space therefore has 2^n possible states, which can be thought of (if not actually visualized) as the corners of a n-dimensional hypercube; I will usually refer to it as a *binary vector space* or as "2^n-space." The value of n is the only information about the function that is assumed to be available before the computation begins—in the connectionist model presented, it affects the size of the network to be used.

---

2 There are technical considerations, discussed later in this chapter, concerning what kinds of penalty terms are generally most effective.

There are a number of reasons for this choice of representation. Given the kind of connectionist machines that I wanted to explore, a bit vector representation can be exploited very naturally. Also, since there are only two choices on each dimension, the representation allows studying relatively high-dimensional functions without creating spaces so large as to be intractable given the amount of computational power available. To say it another way, when spaces grow as 2^n rather than, say, 100^n or $1{,}000{,}000^n$, larger values of n can be investigated. For a given size state space, a hypercube organization possesses the maximum number of non-trivial dimensions.

The black box model might lead one to imagine there could be some single system that could, in a reasonable amount of time, optimize any function fitting the input/output constraints; alas, the combinatoric explosion ensures this is not so. Every time n increases by one, the number of possible hiding places for the maximum doubles. There exist "maximally unhelpful" functions , such as the class of functions returning value zero for $2^n - 1$ of the possible inputs, and value one for the remaining input. Without advance information about such functions, no possible search method could expect to find the maximum before checking about half of the entire space. Such functions cannot be optimized without inside information; they must be avoided.

Unfortunately, such worst case functions occur naturally in some kinds of computations; the n-input boolean AND function is a canonical example. Boolean functions in general are problematic for this formulation, since the search difficulty posed by a boolean function depends only on the number of inputs returning value one. After all, the search terminates the first time a point that makes the function true is found. All of the points tested in the search, except for the last one, score zero. The only information about nature of the function that zero-valued points supply is that those specific points are not maximal. In large spaces, such points say essentially nothing about what other, untested, points *might be* maximal. If the function is highly disjunctive, goal states will be common, and any search method can do well; if the function is highly conjunctive, goal states will be rare, and no method can.

1.2.1 The methodology of heuristic search. It is important to be clear about what the black box function optimization problem implies for motivating and analyzing search strategies. If there is *a priori* information available about the functions-to-be-optimized, it is sometimes possible to *derive* a strategy that is perfect for the problems. As a trivial example, if it is given that the functions are linear , there is an obvious hillclimbing strategy that guarantees to optimize the function in no more than $n + 1$ function evaluations (see Chapter 3). Of course, limiting the problems to linear functions is a very big restriction; the relevant point here is that *given* the functions are linear, there is a *provable reason* why hillclimbing is the appropriate choice of search strategy for binary spaces.

With the black box problem formulation, no such information about the functions is available. This does not mean that using hillclimbing (or any

particular search strategy) is necessarily a bad idea, but it does mean that it is extremely difficult, if not impossible, to prove that a search strategy is optimal. Optimal with respect to what? Hillclimbing simply *assumes* that the function is unimodal, but that assumption may or may not be true of any particular function that the strategy is called upon to optimize. Similar statements hold for other search strategies, though it may be difficult to express clearly the implicit assumptions behind any given search strategy. In the black box problem formulation, with all possible functions as potential problems to be solved, *all search techniques are heuristic.*

From one point of view, this is an unfortunate state of affairs. As a methodology, artificial intelligence research frequently *begins* with limiting assumptions about the possible problems, and then seeks strategies that exploit those assumptions effectively. But in the black box formulation, on what basis is a search strategy to be designed? How is one supposed to get a handle on "all possible functions"? Any *a priori* assumptions will be falsified by some possible function, so does this mean that any search strategy is as good as any other?

Of course not. Even if we cannot give a strategy that solves all problems efficiently, that doesn't mean that all strategies are equally useful. Rather than making assumptions about the problem, we can work from the assumptions made by problem-solvers. For example, hillclimbing implicitly assumes that functions are unimodal. If it searches a multimodal function, its assumption is invalid, and it may get stuck on a local maximum. By contrast, consider an *iterated hillclimber* that starts from a random point, hillclimbs until all further moves lead downhill, and then starts over from a new random point. This strategy can search unimodal spaces as fast as a simple hillclimber, but it can also search certain multimodal spaces effectively (as discussed in the next section). We can conclude that iterated hillclimbing is equally efficient but more general than simple hillclimbing.

This little example illustrates an alternative methodology for approaching the design of search strategies. Rather than beginning with assumptions that limit the set of possible functions, and trying to exploit those assumptions effectively, we can begin with assumptions that existing search strategies make, and try to relax them. To do this, it is very important to make the assumptions underlying a search strategy as clear as possible. It is also very important to perform empirical comparisons of search strategies, despite numerous obstacles making it difficult to produce a "level playing field" for the comparison.

1.2.2 The shape of function spaces. High-dimensional function optimization problems can arise in all sorts of contexts, from physics to economics to artificial intelligence, and a solid intuitive understanding of their nature could be of great value. Unfortunately, high-dimensional spaces are notorious for their counter-intuitive properties. If we are looking for intuitive characterizations, about the best we can do is consider various low-dimensional visualizations or projections of high-dimensional spaces. Such projections reveal certain aspects

of the space while distorting others. For example, the hypercube characterization encourages us to visualize a cube, which emphasizes the fact that from any corner we can reach n other corners by moving across one edge, but suppresses the fact that at each corner there is an associated function value that we are trying to maximize. We can improve this somewhat by imagining varying sized "blobs" at each corner that represent function values by their size. Or, we can imagine all of the states laid out in the xy-plane, and envision a surface in the z-dimension specifying the associated function values. One difficulty in that case is that the extreme connectivity of the high-dimensional space is suppressed—it looks as though some points are widely separated from each other, whereas in fact any point can be reached from any other in no more than n single bit moves. It can be taken as (perhaps sad but) axiomatic that *there is no completely satisfactory way to visualize arbitrary high-dimensional spaces.*

Given this state of affairs, and the discussion in the previous section, one way to go about categorizing high-dimensional spaces is not directly in terms of what they "look like," but indirectly, in terms of what search techniques could reasonably be expected to optimize them. Similar sorts of search difficulties can occur in both coarse-grained high-dimensional spaces and fine-grained low-dimensional spaces, such as "false peaks" that lead hillclimbers away from solution states, and flat "plateaus" that offer no uphill direction to follow. This makes it possible to create easily visualized, low-dimensional spaces that suggest, albeit imperfectly, the kind of difficulties that a high-dimensional function optimizer is likely to face, and the basic reasons why various search strategies are likely to succeed and fail in different situations. The remainder of this section builds a loose taxonomy of function spaces based on this idea, and provides low-dimensional analogues to demonstrate the qualitative characteristics desired. The spaces start out easy and become harder, in the sense that a strategy suitable for a given class will usually be able to handle any of the previous classes, although perhaps less efficiently than those strategies that are ideally suited for the easier case.

Remember that these two-dimensional examples are just "visual aids," and they are *not* the functions that are actually tested in the later chapters. Several circumstances where the low-dimensional analogues are misleading are pointed out along the way. Issues arising from binary vector spaces are considered at greater length in the following section, and Chapter 3 provides high-dimensional sample functions possessing the various characteristics discussed here.

Figures 1–2 through 1–9 diagram "landscapes" of several functions defined on the 2-dimensional xy-plane, with z representing function value. x and y values run from -10 to $+10$ in all plots. To accommodate the discrete state space assumption, we can assume there exists a 100×100 grid of 10,000 sample points. Of course, it would not take that long simply to evaluate all ten thousand points; to understand these examples, we have to pretend that only dozens or perhaps hundreds of evaluations can be performed in feasible time. Figure 1–2 displays an example of the most benign sort of spaces one may encounter—spaces in

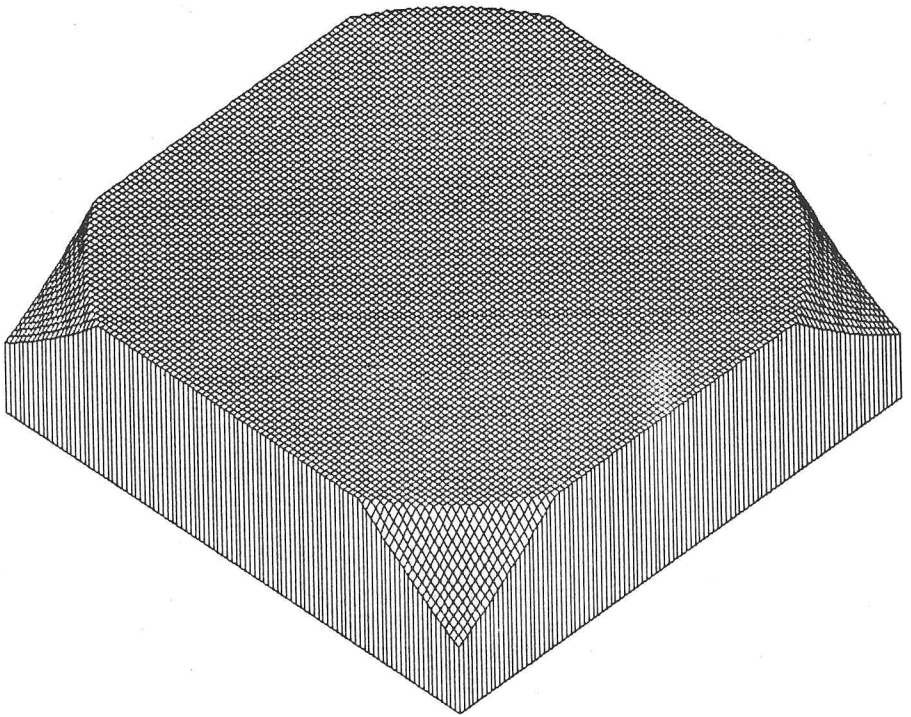


Figure 1–2. A surface plot of a mostly maximal function.

which a high percentage of the states possess the maximal value. In this sort of "mostly maximal" space, simply picking points at random will succeed very rapidly.

Figure 1–3 shows a linear function . A globally maximum value can always be found at one or more of the "corners" of the space: $(-10,-10)$, $(-10,10)$, $(10,-10)$, or $(10,10)$. There are no surprises in a linear function; in the two dimensional case, extrapolating from three non-collinear points anywhere on the surface will identify a maximal corner.

Figure 1–4 shows a non-linear function, $z = 200e^{-.2\sqrt{x^2+y^2}}$. Extrapolation from a few points is no longer sufficient to find the global maximum. However, this function is *unimodal*: There is only one point that is higher than all points immediately adjacent to it, so that point must be the global maximum. Consider a simple hillclimbing algorithm: Start at a random point; keep moving from the current point to any higher adjacent point; when there are no higher adjacent points, stay there forever. In a unimodal space, such a hillclimber will reach the global maximum without fail, started from anywhere in the space. All locally uphill moves are part of a globally uphill path to the summit.

Although simple hillclimbing is guaranteed to work in unimodal spaces, it

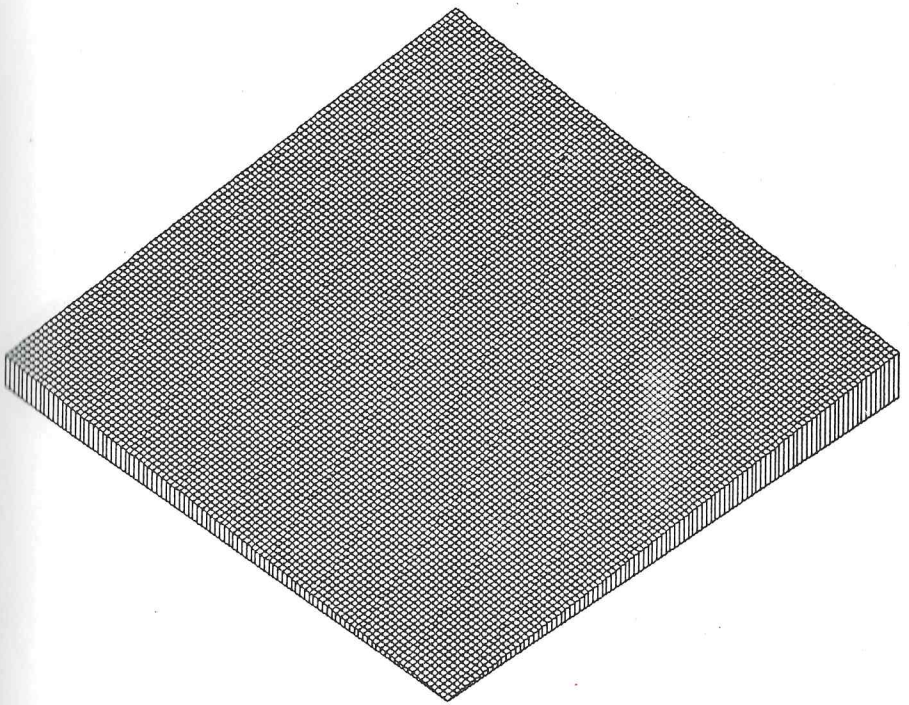


Figure 1–3. A surface plot of a linear function.

is worth pointing out that the path to the solution need not always be very direct. In Figure 1–4, for example, suppose there was a serpentine ridge in the landscape, spiraling around and around and up the side of the mountain. The hillclimber would get "caught" on the ridge, and the locally uphill path would lead it on a merry tour around the mountain. The hillclimber would gain altitude only very slowly.

Note also that in binary vector spaces, linear functions and unimodal functions are very similar with respect to search strategies. The extrapolation technique usable in the fine-grained linear case does not help in a binary vector space—to determine which direction to move along a dimension, points must be sampled at two different values, but then there are no untested values on that dimension to extrapolate to. In binary vector spaces, the primary difference between linear and unimodal functions has to do with the possible lengths of uphill paths in the space. With a linear function, there is no reason to ever consider changing a bit that has already been considered before—if, at the current point, the function value with bit 6 turned on is higher than with bit 6 turned off, then that will also be true at every other point. The longest possible uphill path in a linear binary vector space of n bits is of length n, and occurs when the starting point is the bit-wise complement of the solution state.

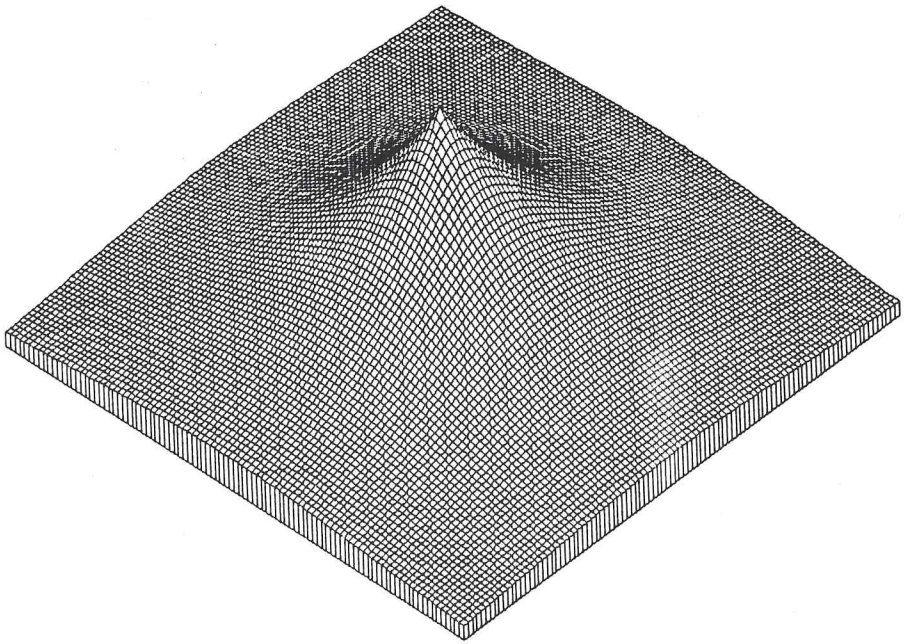


Figure 1–4. A unimodal function, susceptible to simple hillclimbing.

With unimodal functions, it is possible to force a hillclimber to follow a path that is longer than the dimensionality of the hypercube. For example, at the current point it might improve the function value to turn bit 6 on, but further down the path it might be better still to turn it back off again. Such situations are high-dimensional analogues to the ridged mountain discussed above—at some points it is better to head in the positive x direction, but later on it is better to do the opposite. In any event, since extrapolation means little in binary vector spaces, we can view the unimodal case as subsuming the linear case, and we can take "unimodal" as the second category of functions, with "simple hillclimbing" as the simplest applicable search technique.

Now consider Figure 1–5. This function is multimodal, so simple hillclimbing cannot be relied upon to find the global maximum—if it happens to start out somewhere in the western corner of the space, it will climb the lower peak, get stuck, and fail to maximize the function. For this reason, multimodality is sometimes taken as *prima facie* grounds for rejecting a hillclimbing approach; this landscape is included to show that the situation is not always that simple. An *iterated* hillclimbing algorithm —one that hillclimbs from a random point until no further improvment is possible, then starts over—could do quite well here. The sides of the two pyramids are equally steep, and therefore the higher pyramid must also be wider at the base. With every random choice of start-

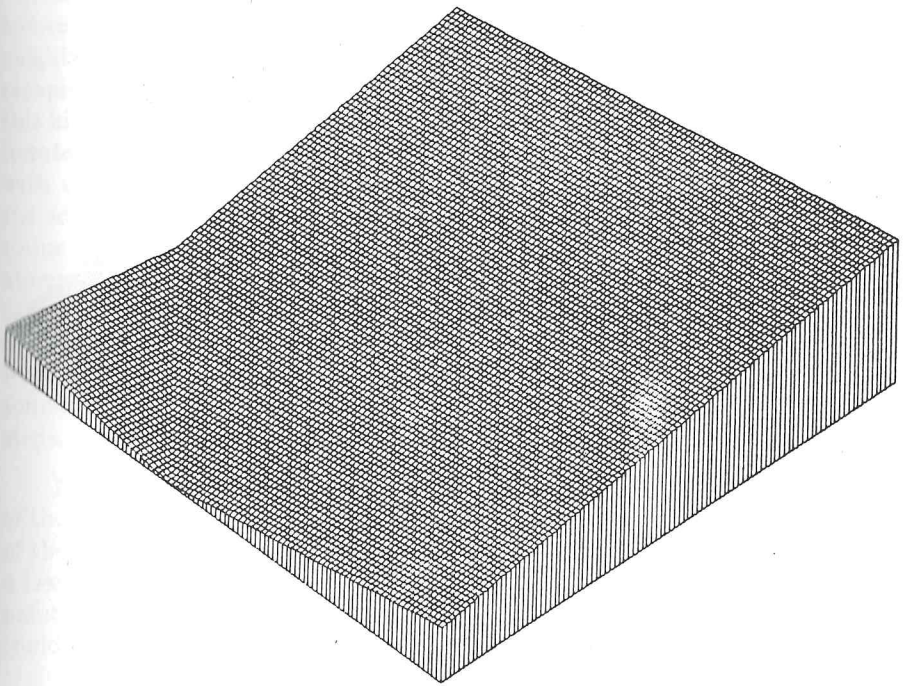


Figure 1–5. A largely-unimodal function, susceptible to iterated hillclimbing.

ing point, an iterated hillclimber is odds-on to start somewhere on the higher pyramid.

If we look at this landscape upside down and consider minimizing rather than maximizing, we can get a particularly intuitive picture. If we imagine each iteration of the hillclimber as being a raindrop that lands somewhere on the surface and then runs downhill, the area covered by each valley—the "watersheds" or "collecting areas" of the space—takes on obvious significance. In particular, we can see that the size of the watershed surrounding the global minimum—compared to the size of the space—is the critical factor is determining the success of iterated hillclimbing. For "largely unimodal" spaces such as this, iterated hillclimbing will do well.

Figure 1–6 displays a more complicated function, $z = 200e^{-.2\sqrt{x^2+y^2}} + 5e^{\cos 3x+\sin 3y}$. Broadly speaking, this landscape looks much like Figure 1–4, but the second term in the equation imposes many small hills and valleys on the broad exponential mountain. The collecting area of the global maximum is small, and an iterated hillclimber will get stuck on one of the hilltops most of the time. However, there are other local search methods that can do quite

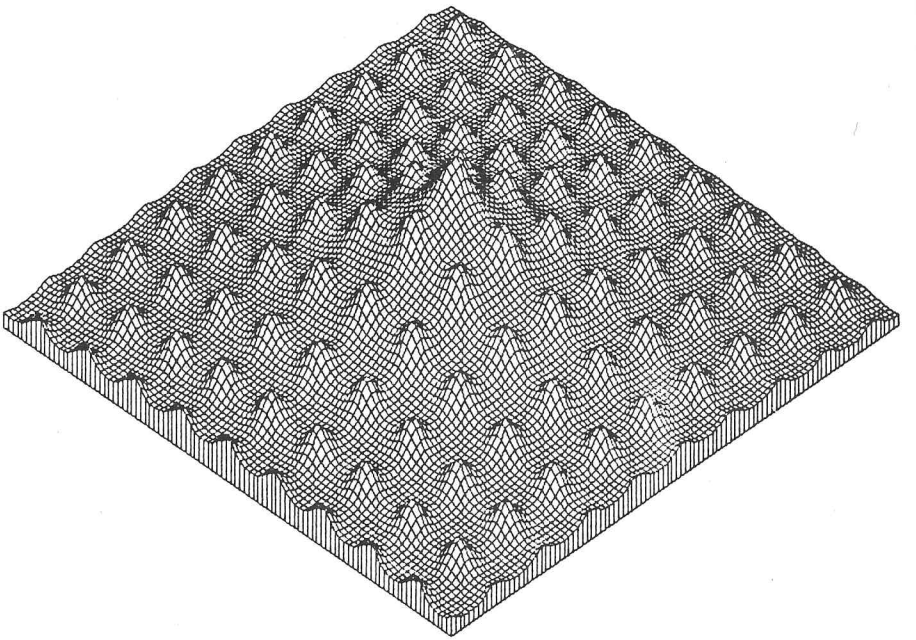


Figure 1–6. A fine-textured broadly unimodal space problematic for hillclimbing but susceptible to other local search methods.

well. To begin with, suppose that a search strategy considered all of the points in a "neighborhood" surrounding the current point, instead of only considering immediately adjacent points. Once the size of the neighborhood gets a bit bigger than the collecting areas of the small hills, the search strategy will be able to "see" the higher lands hidden by the intervening valley,[3] and will be able to move through the valleys to higher and higher hills.

Such neighborhood search procedures have the effect of smoothing local irregularities in the landscape, effectively removing local maxima that are sufficiently small with respect to the size of the neighborhood. The price for this service, though, is the increase in the number of function evaluations per move. In low-dimensional cases, one might be able to consider fairly large neighborhoods without becoming excessively slow, but this trades on the limited connectivity of low-dimensional spaces. As the dimensionality of the space grows linearly, the number of neighbors grows exponentially, and the number of function evaluations required to evaluate even a modest neighborhood grows rapidly.

---

3 If we imagine the hillclimber is searching while clinging to the *underside* of the surface, rather that than standing on top of it, then valleys *will* hide mountains.

An alternative perhaps more obviously suited to high-dimensional spaces is a *stochastic hillclimbing* search strategy. We do not have to check all neighbors before deciding upon a move; we can make that decision after checking each neighbor. The idea is to sometimes go downhill as well as up, with the hope of escaping from local maxima in the process. There are many ways to implement this kind of procedure. For example, start at a random point in the space, then iterate the following: Consider a random adjacent point and compare its height with the current point. Call the difference in height d, with $d > 0$ implying the adjacent point is higher. Now make a probabilistic decision about whether to move to the adjacent point or to remain at the current point. The decision about whether to accept such a move is made based on d and on a parameter called the "temperature."[4] At zero temperature the procedure will accept a move if and only if $d > 0$. At infinite temperature it will accept any move with a probability of 50%, regardless of d. At intermediate temperatures it will sometimes go up and sometimes go down, preferring uphill steps over downhill steps, and smaller downhill steps over bigger ones.

Note that on this landscape, the base of each hill is highest in the direction of the mountain top. This means that if a stochastic hillclimber is stuck on one of the hills, it has a greater chance of escaping to a higher adjacent hill than to a lower one, since fewer downhill steps are required to reach the higher saddle point. At a temperature appropriate to the landscape, a stochastic hillclimber could do fairly well here.

There is a sense in which this is rather inefficient search procedure, since it is perfectly possible that, after taking one or more downhill steps away from a local maximum, the procedure will randomly turn around and climb right back where it came from, accomplishing nothing. In low-dimensional spaces such as Figure 1–6, this can be a significant problem. However, the situation is somewhat different in binary vector spaces. At each point, there are n choices of where to try next. As n gets larger, the chances of exact backtracking become very small, and the number of possible places the procedure could reach in a few steps grows very rapidly. Suppose a stochastic hillclimber takes one downhill step away from a high-dimensional local maximum, and the situation is such that three downhill steps are required to escape. Each time it considers a random adjacent point, it has only a $1/n$ probability of picking the point it just vacated. Except when n is small, the hillclimber will end up having on average several chances to accept another downhill move, and thus get further away from the local maximum, improving its chances of escape.

Of course, the flip side of being good at escaping local maxima is being bad at climbing to the top of the global maximum. When there are lots of downhill moves being accepted, there is much less pressure to get to the top of any given hill. One way to deal with this problem is to vary the temperature during the course of the search. Such an idea underlies the *simulated annealing* technique

4 This procedure is formalized in Chapter 3.

(Kirkpatrick, Gelatt, & Vecchi 1983; Cerny, 1985). Simulated annealing is a stochastic hillclimbing algorithm that starts out at high temperature and gradually lowers it as time passes. On the landscape in Figure 1–6, a simulated annealing algorithm at very high temperature would wander all over the place, then spend more and more of its time concentrating on the central peak as the temperature was lowered. Intuitively speaking, this is a simple example of the sort of space for which simulated annealing is ideally suited. The space has local maxima, so simple hillclimbing is not enough, and the collecting area of the global maximum is small, so iterated hillclimbing will be slow, but the problem can be solved "outside in"—from the big picture down to the fine details. What defines the "big picture" in this case is distance on the xy-grid. If you look at Figure 1–6 from increasingly far away, or increasingly out-of-focus, the small local hills become less and less visible, and the overall mountain shape becomes more and more dominant. Such operations have the effect of low-pass filtering the space, so that local irregularities are smoothed out, and only the broader, "low frequency" components of the landscape remain. Running at an appropriate temperature has much the same effect in terms of the behavior of the search process over time: the probability of taking enough downhill steps to escape a local hillock is quite high, but the probability of wandering all the way down the overall mountain is quite low. In such "fine textured broadly unimodal spaces," stochastic hillclimbing techniques can perform very well.

In more seriously convoluted spaces, local methods begin to break down. For example, Figure 1–7 displays the landscape generated by the function $z = e^{-.2\sqrt{x^2+y^2}+3(\cos 2x+\sin 2y)}$. In the previous space, the local hills and valleys were small relative to the height of the landscape, but now they are the dominant features of the landscape, and the global maximum is just a hilltop somewhat higher than the rest. The bases of the hills are at almost the same height in all four directions, compared to the height of the hills, and except at the edges the number of grid points covered by each hill is the same. In situations like this, local search methods run into serious trouble, because their basic assumption is violated: The locally uphill directions almost never point to the global maximum. A hillclimber would almost certainly climb a suboptimal spike and get stuck. A simulated annealing algorithm would have to start at a high temperature to avoid the same fate, but then downhill moves would be common and there would be little pressure to climb all the way up the optimal spike. As the temperature is lowered, a "critical point" is reached at which the probability of moving from one spike to another becomes very low, and the algorithm will then concentrate on reaching the top of whatever spike it is on. Unfortunately, on this landscape the critical temperature is so high, relative to the differences in the heights of the spikes, that the chance of ending up on the globally optimal spike is very small, unless the system is cooled *extremely* slowly through the critical region.

Nonetheless, there is an obvious structure to the landscape. The positions and heights of the hilltops are not random. There are 10,000 sample points, but there are only 49 hilltops, 7 of which have the same x-coordinate as the global

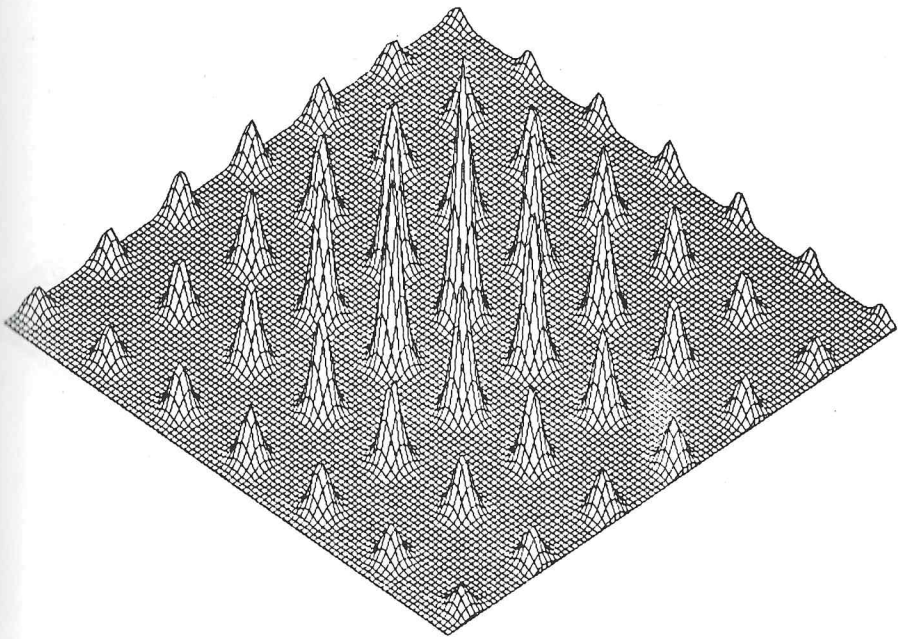


Figure 1–7. A coarse-textured broadly unimodal space problematic for local methods but susceptible to global search methods.

maximum, and 7 of which have the same y-coordinate as the global maximum. So far, local maxima have been viewed solely as obstacles to be avoided, but in spaces such as this *the coordinates and heights of the hilltops contain useful information.* The fact that a simple hillclimber will quickly get stuck on a local maximum, which had previously been viewed as a drawback, can be turned to advantage in this circumstance.

A more complex algorithm could use simple hillclimbing as a single step in a global search process. For example, an algorithm could run a number of hillclimbs from random points, and record the locations and altitudes of the discovered local maxima for future use. Then instead of choosing starting points for further hillclimbs at random, a starting point could be chosen by *combining* the x coordinate of one discovered local maximum with the y coordinate of another.[5] Furthermore, if the choice of local maxima is biased towards the higher valued known points, a natural pressure will exist driving the algorithm towards considering points that lead uphill to the overall global maximum.

The new search technique proposed in this dissertation works basically in

[5] The assumptions about the shape of the search space embodied by this rule are discussed in Chapter 2.

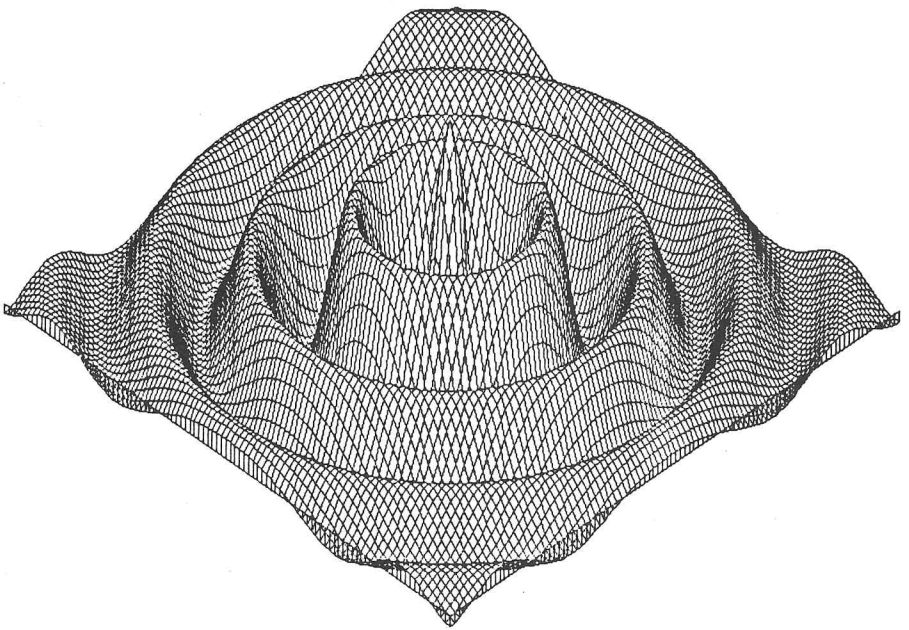


Figure 1–8. A globally structured space problematic for genetic hillclimbing.

that fashion. I call the procedure "stochastic iterated genetic hillclimbing," (SIGH) because it combines the basic elements of genetic algorithms—a population of points, a combination rule for generating new points, and a mechanism for providing higher valued points with more reproduction opportunities—with the efficient local optimization abilities of hillclimbing algorithms. It is called "iterated" because the combined genetic/hillclimb process typically occurs a number of times in the course of the search for the optimum value of a complex function. It is called "stochastic" because the timing of the shifts back and forth between genetic search and hillclimbing are determined, ultimately, by random variables, rather than occurring according to some fixed control regime.

High-dimensional analogues of the landscape in Figure 1–7 are the central concern of Chapter 5, and SIGH proves to be a formidable search strategy for some of those functions. A related algorithm, which I call "iterated genetic hillclimbing" (IGH), is also tested in Chapter 5. It possesses some of the properties of SIGH, but is less expensive to perform on a sequential machine. It can be viewed as an iterated hillclimber in which the starting points are chosen by genetic recombination, or as a genetic algorithm in which points generated by recombination are "improved" by hillclimbing before being added to the gene pool. IGH is just a simple-minded combination of genetic algorithms and hill-

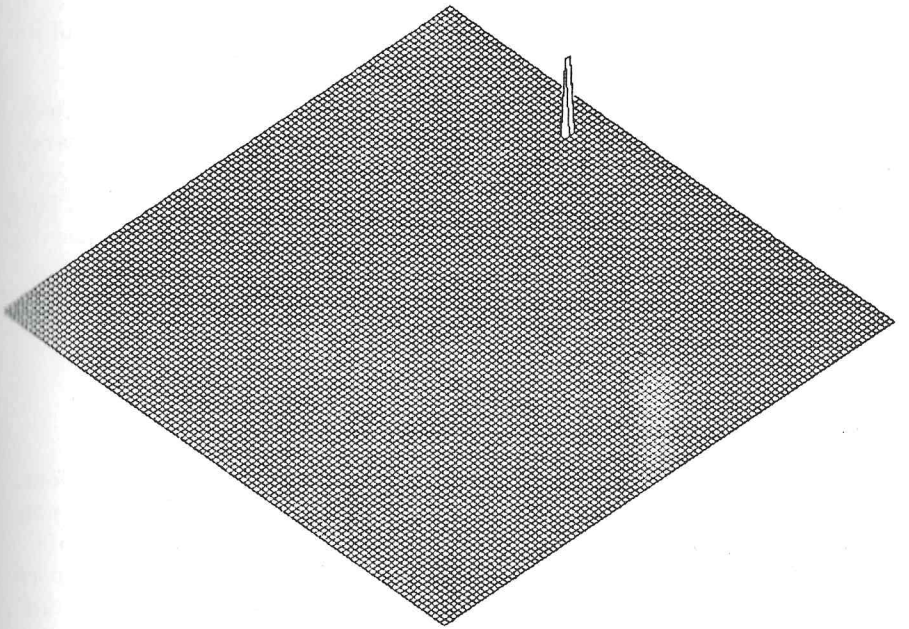


Figure 1–9. A conjunctive boolean space problematic for any search procedure.

climbing, but it turns in by far the fastest performance on the hardest problem tested in Chapter 5.

It should be noted that although such global techniques can succeed in cases where strictly local hillclimbing methods fail, there are parameter spaces with obvious structure that will foil these techniques as well. The landscape in Figure 1–8 is an example. The x and y coordinates of the local maxima and the global maximum only rarely and accidently match closely enough to be helpful. Genetic techniques cash in on such correspondences, but there is no guarantee that they will exist.[6]

Finally, to complement the essentially trivial function in Figure 1–2, Figure 1–9 plots an essentially impossible function $z =$ if $(x = -3.2$ and $y = 8.7)$ then 100 else 0. Any reasonably sized sample of the space would lead one to the conclusion that it is a completely flat linear function with maximum value

6 It is interesting to note that in this particular case, if a non-linear transformation — translation to polar coordinates—is performed, the resulting landscape is significantly easier to search, as it becomes one-dimensional. Such non-linear changes of representation can be extremely powerful tools, but their use is beyond the scope of this dissertation, which focuses on doing as well as possible in a fixed representation.

zero; stumbling across a non-zero value would be most unexpected. Conjunctive spaces like this are simply intractable for black box optimization, regardless of what search technique is used, as the global maximum provides no hint of its existence unless its exact combination of parameters is guessed.

It must be stressed that the classes of functions discussed—largely maximal, unimodal, largely unimodal, fine-textured broadly unimodal, coarse-textured broadly unimodal, and intractable —are only stereotypical "essences," and they do not begin to exhaust the possible shapes of function spaces. This particular breakdown is organized on the basis of "searchability," which has the advantage that if an arbitrary function space falls more or less squarely into one of the categories (except intractable), an *a priori* reasonable search strategy is immediately suggested.

1.3 High-dimensional binary vector spaces

The functions depicted in the previous section were defined on two dimensions, with each dimension possessing 100 distinguishable values. This made it possible to display the landscapes in a visually coherent form, but it is uncharacteristic of the functions actually considered in the thesis. An n bit vector space has n dimensions, with each dimension possessing two distinguishable values. There were $100^2 = 10{,}000$ states in the two dimensional examples; a comparably sized binary space might have thirteen dimensions with $2^{13} = 8{,}192$ states. With two fine-grained dimensions, points in the space can be represented as ordered pairs, like $(-3.2, 8.7)$; with thirteen binary dimensions, points can be represented as thirteen-bit vectors, like $(1,0,0,1,1,1,1,0,1,1,0,0,0)$, or—to save space—1001111011000.

As a notational matter, henceforth in this thesis the binary values of zero and one will be denoted by '*0*' and '*1*', rather than by '0' and '1'. The reason for this is that the proposed model is designed to optimize functions defined on the hypercube with corners at $+1$ and -1, rather than at 0 and 1. To allow it to search the more standard unit hypercube, and to allow discussion of the model in *0/1* terms, a translation is performed, mapping -1 onto *0* and $+1$ onto *1*. It is important to keep the two interpretations of the binary values separated, because the proposed model employs 0 values for a different purpose, so 0's *do not* correspond to *0*'s. The symbols *0* and *1* should not be taken as representing numbers, *per se*, but as representing the abstract alternative outcomes for a two-valued decision. The thirteen-bit vector above represents a sequence of thirteen two-valued decisions, so by this convention, it should be presented as *1001111011000*.[7]

[7] Given that I have squeezed the punctuation out of the binary vector representation, another reason for adopting this notation is to help resist interpreting binary *vectors* as binary *numbers*. The function-to-be-optimized is responsible for assigning meaning to each of the decisions in a binary vector, and there is no *a priori* interpretation of *0* and *1* as numbers, let alone ideas like "most significant bit" or "least significant bit."

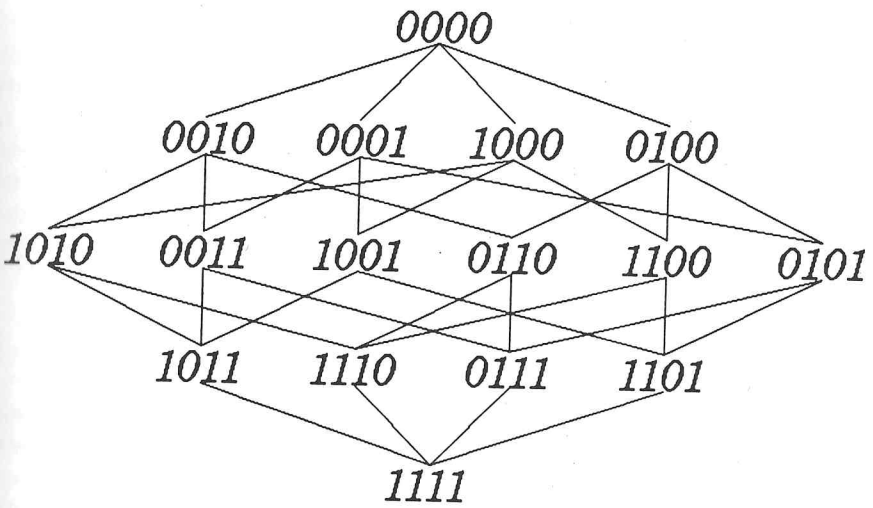


Figure 1–10. Paths through hamming space from *0000* to *1111*.

Using many short dimensions instead of few long dimensions has a number of important consequences. Perhaps the most unintuitive change is that the concept of *direction* in the space loses most of its meaning. In spaces with fine-grained dimensions, one can pick a heading—such as two steps up on one dimension for each one step down on another dimension—and navigate a straight course, as the crow flies, for many steps. Motion in a binary space is much twistier. The smallest step is from one end of a dimension to the other end. It is not possible to continue further in that direction; after each step, a new direction must be chosen.

Although direction does not mean much in a binary space, *distance* still has a sensible interpretation. The *hamming distance* between two points in a binary space is the number of bit positions that do not match. If a search algorithm changes only one bit per time step, the hamming distance is also the minimum time to reach one pattern started from the other. The lack of direction appears again: the shortest path between two points is not unique. There are many equally direct paths between distant points, depending on the order in which the intervening dimensions are traversed. Figure 1–10 shows the possible direct paths between the two points *0000* and *1111* on a four dimensional binary space. Each layer corresponds to one unit of hamming distance from the layer above and below. The number of possible intermediate states grows in each layer until the midpoint is reached, at a hamming distance of two from each of the endpoints. Since the two points are complements of each other, points that are halfway across hamming space match the endpoints exactly as often as they mismatch.

The black box model assumes the function will evaluate any point on demand, so the next point evaluated can be any distance from the previous point

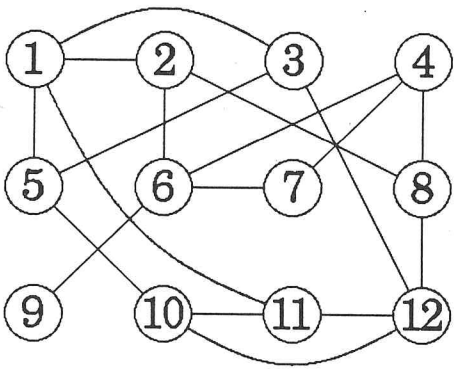


Figure 1–11. A graph with twelve nodes and eighteen edges. The minimum cut partitioning problem for this graph is to divide the twelve nodes into two groups of six such that the number of edges connecting nodes in different groups is minimized.

evaluated, should there be a desire to do so. Hamming distance is useful as a crude measure of similarity between points, representing the weak assumption that all of the dimensions are equally important in determining similarity.

Small binary spaces can be visualized. A one dimensional space corresponds to the ends of a line segment; a two dimensional space corresponds to the corners of a square; a three dimensional space corresponds to the corners of a cube. Picking a random three bit vector selects a random corner of the cube. A local move travels along an edge from one corner to another, corresponding to flipping one bit in the vector. Non-local moves change more than one bit at a time, and move between corners that are not directly connected by an edge. In this picture, all three dimensions are used to represent the parameter space, so the function value cannot be visualized as "height." One could imagine boxes at the corners of the cube, each box containing a piece of paper with the corresponding function value written on it.

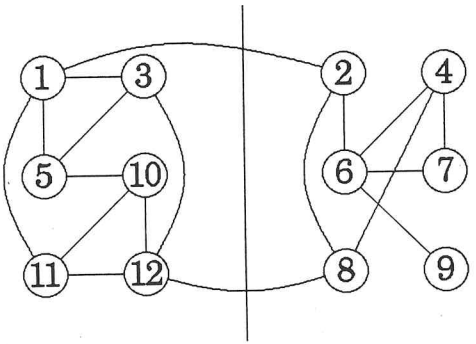
1.3.1 Graph partitioning. Beyond three dimensions, visualization in spatial terms becomes difficult. It is hard to imagine what so many different dimensions could mean. An example may help illustrate the possibilities for interpretation. Consider a problem called *minimum cut graph partitioning* (Garey & Johnson, 1979; problem ND17: Minimum cut into bounded sets), which is explored at length in Chapter 5. A simple form of the problem begins with a graph consisting of a set of nodes and a set of edges connecting

the nodes. Figure 1–11 shows a graph with twelve nodes and eighteen edges connecting them. The problem is to separate the nodes into two groups of six nodes each, such that the number of edges that cross between the two groups is as small as possible. One way to look at it is that one is to color half of the nodes black and half of the nodes white, such that the number of links between different colored nodes is minimized. Another way to look at it is that one is trying to chop the graph into two separate equally-sized pieces by cutting as few edges as possible. For example, the (non-optimal) partitioning $\{1,2,3,4,5,6\}$ and $\{7,8,9,10,11,12\}$ produces a "cut size" of eight edges.

A place where this sort of problem arises is when a circuit is too large too fit on a single chip (or printed circuit board, or whatever the unit of granularity happens to be), so it must be placed on two units with a minimum number of interconnections between the units. Graph partitioning is an obvious choice in this case. Nodes in the graph represent devices and edges represent wires connecting the devices. However, graph partitioning also finds uses in less obviously related contexts. For example, Dunlop & Kernighan (1985) use graph partitioning to lay out "standard cell" circuits on a single silicon chip. Graph partitioning is used to answer this question: How should the various circuit elements be laid out on a chip so that the amount of wire needed to interconnect them properly is minimized? Tightly coupled nodes should be placed nearer each other, and loosely coupled nodes can be placed farther apart. By partitioning the circuit graph, and recursively partitioning the partitions, and so forth, a general "closeness metric" is derived for the nodes of the graph, which the layout algorithm uses to arrange the circuit elements on a chip efficiently.

Figure 1–12a shows a minimal partitioning of the graph. The two groups of nodes, $\{1,3,5,10,11,12\}$ and $\{2,4,6,7,8,9\}$, are only connected by two edges. A partitioning can be compactly represented as a bit vector as shown in Figure 1–12b. A bit is used for each node, with zero or one indicating which side of the partition the node is to be placed. The constraint that there be equal numbers of nodes on both sides of the partition implies there must be an equal number of zeros and ones in the vector. It doesn't matter which group of nodes goes on which side of the partition, so an equivalent solution is *101010000111*.

With such a representation, the dimensionality of the space is the number of nodes in the graph being partitioned. However, not all of that space is directly relevant to solving the problem. In this example, there are $2^{12} = 4{,}096$ states in the space, but only $\binom{12}{6} = 924$ of those states are *balanced*—having equal numbers of zeros and ones—as required by the problem. The function must be defined over the entire parameter space, including the unbalanced states, to apply black box function optimization. In some sense an unbalanced solution is no solution at all, since the constraint is part of the problem statement. One obvious idea might be to set the function value for unbalanced states to some very high number, greater than the total number of edges in the graph. Since graph partitioning is a minimization problem, this makes every unbalanced state less desirable than any balanced state. Unfortunately, it also leads to a very choppy landscape , of the sort that caused difficulties in the previous



(a)

010101111000

(b)

Figure 1–12. Two representations of a minimal partition of the graph of Figure 1–11. *(a)* A graphical representation of the partition. *(b)* A representation of the partitioning as a binary vector. One bit is used per node, with the leftmost bit denoting node 1 and the rightmost bit denoting node 12. A *0* means "this node left of partition," and a *1* means "this node right of partition."

section. Large regions of the space contain only unbalanced states and would be entirely flat. A local search function minimizer that started in such a region would detect no downhill direction to follow. Furthermore, changing any single bit in the vector changes the balance of the solution, so each balanced state would be entirely surrounded by much worse states. If a simple hillclimber was lucky enough to stumble on a balanced state, it would be trapped there, no matter how bad the resulting cut.

What would be desirable is a smoother notion of balance, one that rewards improved balance but still allows a slightly unbalanced solution with a small cut to score more highly than a perfectly balanced solution with a terrible cut. This can be accomplished by defining the function value of an unbalanced state to be the size of the cut plus a penalty term, where the size of the penalty depends on how unbalanced the state is. With this scheme, an unbalanced region of the space is no longer flat, but instead has a downhill slope towards the nearby balanced states. This is quite useful because it allows good balanced states to be "sensed" from some distance away. For example, consider the partition obtained by swapping node 10 and node 6 in Figure 1–12a. In that state, the cut size is nine. Moving either node back to its original location would significantly improve the cut, but would produce an unbalanced state. If the penalty for being one bit out of balance is less than three (the minimum number of edges removed from the cut by moving node 10 or node 6), there will be two strictly downhill paths back to the minimal partition, even though they pass through unbalanced states along the way.

A drawback to the penalty approach is that the actual minimum of the composite function may not be a balanced state. Although in practice a slight imbalance for the sake of a better cut is often a favorable tradeoff, for the sake of a fair comparison with other algorithms it is necessary to provide a way to guarantee a balanced solution.[8] If an algorithm may converge at an unbalanced state, for whatever reason, one can add a "backstop" hillclimbing step using a very large penalty term just to balance the final state.

Graph partitioning provides an intuitive way to understand the fundamental problem that makes many high-dimensional functions hard to optimize. The optimal placement of a given node, say node 1, cannot be determined in isolation, because it depends on where nodes 2, 3, 5, and 11 have been placed. Each edge between a pair of nodes represents a *non-linear dependency* between the corresponding dimensions. Other things being equal, the best placement for a pair of connected nodes, say node 6 and node 9, is either *11* or *00*, since the other two states *10* and *01* increase the size of the cut. The state of one node by itself has no relation to the cut size; it depends on the states of the nodes it is connected to.

An important factor determining how troublesome such dependencies are is the *order* of the non-linearity. Informally speaking, the order of a non-linearity is the number of bits whose best values must be determined simultaneously. The case of two nodes connected by an edge is called an order 2 dependency, since the best value of two bits must be decided simultaneously. Higher order dependencies can be built out of order 2 dependencies. For example, each edge in a graph partitioning problem introduces an order 2 dependency, and the placement of node 1 in Figure 1–11 involves at least an order 5 non-linearity, since node 1's contribution to the cut size depends simultaneously on its own placement and the placement of the four other nodes it is directly connected to. Furthermore, the placement of those nodes in turn depends upon the nodes to which they are connected, and so forth, leading to a situation in which the best placement of any node depends in varying degrees on the placement of all the other nodes.

In general, the existence of high-order dependencies makes function optimization difficult. The worst-case n-input boolean AND function mentioned previously is of order n. In graph partitioning, a rough measure of difficulty can be found by examining the *degrees* of the nodes—the number of edges emanating from the nodes. If edges are sparse and the average degree of a node is low, moving a single node across the partition will not usually make the cut that much worse. A local approach such as simulated annealing, which is willing to move from one solution to a somewhat poorer one, could do reasonably

---

8 Although in practice, when using an algorithm that guarantees to find a balanced partition, it is common to add some "dummy nodes" that aren't connected to anything, so they can be placed anywhere without affecting the cut. The number of dummy nodes determines how unbalanced a partition of the original graph is tolerable (Dunlop & Kernighan, 1985).

well on sparse graphs. As the graph becomes denser and the average degree of a node gets higher, this becomes more difficult. In particular, if a graph is "clumpy"—containing clusters of nodes that have relatively many edges between themselves but relatively few to the rest of the graph—it becomes less and less likely that a stochastic hillclimber will take enough consecutive uphill steps to move a whole clump across the partition. Moving any one node of a clump across the partition makes the cut significantly worse. Even ignoring balance, about half of the clump has to be moved across the partition, against the gradient, before the slope of the landscape shifts towards favoring moving the rest. A non-local approach could try to identify sets of dimensions that are tightly bound in clumps, and could move clumps across the partition in blocks, corresponding to flipping set of bits simultaneously instead of one at a time.

1.4 Dissertation overview

The goal of this chapter has been to motivate the particular framework within which I have investigated connectionist solutions for solving strong constraint satisfaction problems when the constraints are hidden from the problem solver. Aside from insisting on a strict success criterion for terminating the computation, which implies that computation time must be the primary performance measure, I have set the problem up to emphasize the need for adaptive search, and to facilitate connectionist modelling.

Chapter 2 presents the model. It begins by reconsidering the various search strategies that have been mentioned so far, organized on the basis of their knowledge representations. A short introduction to the connectionist approach to computation is presented, angled towards facilitating the presentation of the model. The notations and equations defining the model are then presented, and the effect of the learning rule is discussed from several perspectives.

Chapter 3 demonstrates the model on a number of functions. Most of the functions are recognizable high-dimensional analogues of functions discussed in this chapter. As might be expected, on the simpler functions, simpler search strategies prove to be most efficient, but as the landscapes become more rugged, the more global search techniques—the genetic algorithm, the simulated annealing algorithm, and SIGH—begin to display their strengths.

With the behaviors of the various search strategies in hand as a guide, Chapter 4 analyzes the proposed model. The analysis views the model as a form of generate-and-test—with a probabilistic generator controlling the instantaneous search behavior, and a reinforcement process that evaluates the search behavior and modifies the probability distribution used by the generator. The key notion of a *similarity measure*, which determines how the learning generalizes from one point in the space to others, is motivated, and then the specific metric embodied in SIGH is derived. Although complex non-equilibrium effects make a complete formal analysis of SIGH's behavior very difficult, limited analysis of certain special cases is feasible. The model displays two distinctly

different modes of behavior—local search and global search—even though such behavior was not explicitly "programmed in" and was in fact unanticipated before the model was first simulated. An analysis of SIGH as a stochastic hillclimber is given, and steps toward an analysis of SIGH as a genetic algorithm are discussed.

Landscapes such as Figure 1–7 are of primary interest in this dissertation, since in some sense they are near the limit of spaces searchable by black box techniques. Chapter 5 applies SIGH and a number of other search strategies to a series of functions designed to present high-dimensional versions of this problem. As discussed in this chapter, the general framework taken is the graph partitioning problem: Given an arbitrary graph, divide the nodes into two equal-sized groups such that the number of edges between the groups is minimized. Graphs with extensive "clustering" or "clumping" of nodes present difficult partitioning problems—high-dimensional analogues of Figure 1–7—and Chapter 5 presents two such graphs. The basic hillclimbers do not fare well on the larger of these graphs, and the more global methods display differing areas of relative strength and weakness.

Chapter 6 discusses existing work related to the model. In functional terms, the model is related to several approaches to search and learning; in architectural terms it is related to several connectionist research efforts.

Chapter 7 criticizes the dissertation research on a variety of fronts, then speculates on possible variations and extensions to SIGH.

Chapter 8 contains discussion and conclusions. Some of the metaphors and architectural goals that lead me to SIGH are presented. A short discussion of the contributions of the work concludes the dissertation.

1.5 Summary

When there is a strict criterion characterizing the solution points of a space, resource usage—particularly time—is the only relevant measure for comparing the performance of algorithms for searching the space. By hiding the criterion and the structure of the space from the search strategy, a situation is created in which *learning while searching* and *sustained exploration* are prominently emphasized. Intuitively, an ideal algorithm would "learn from its mistakes" (i.e., *all* the points it evaluates, except the last), and would never get "jammed up" by permanently searching in a small region of the space.

Given this problem formulation, random search is an efficient technique only when the search space is mostly maximal. For linear and unimodal spaces, simple hillclimbing is effective (even though it is a convergent technique, unlike random search). For largely unimodal spaces, iterated hillclimbing is effective, and will not waste a great deal of time climbing local maxima.

When there are many local maxima, and the collecting area of the global maximum is small, the simple strategies become inefficient or inapplicable.

When the space can be viewed as the sum of a simple space and a low amplitude "texture" of local maxima, stochastic hillclimbing techniques such as simulated annealing can be very effective. (There are less restrictive definitions of the spaces for which simulated annealing is effective, for example, see Section 6.2.4.) As the texture becomes higher amplitude with respect to the simple space, black box search techniques run into more and more difficulty. When a space is such that there are regularities in the landscape that are aligned with the coordinate axes, "genetic" techniques that generate points by combining coordinates of previously searched points may be effective. Many search spaces are intractable—either currently or inherently—for black box techniques.

References

Ackley, D.H., & Berliner, H.J. (1983). The QBKG system: Knowledge representation for producing and explaining judgments. Technical report CMU-CS-83-116, Carnegie Mellon University, Pittsburgh, PA.

Ackley, D.H. (1984, December). Learning evaluation functions in stochastic parallel networks. Unpublished thesis proposal, Carnegie Mellon University Department of Computer Science. Pittsburgh, PA.

Ackley, D.H., Hinton, G.E., & Sejnowski, T.J. (1985). A learning algorithm for Boltzmann Machines. *Cognitive Science, 9*(1), 147–169.

Ackley, D.H. (1985). A connectionist genetic algorithm. In J. Grefenstette (Ed.) *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* 121–135. Carnegie Mellon University, Pittsburgh, PA.

Alspector, J., & Allen, R.B. (1987). A neuromorphic vlsi learning system. In P. Losleben (Ed.), *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference.* Cambridge, MA: MIT Press.

Barto, A.G., & Sutton, R.S. (1981). Landmark learning: An illustration of associative search. *Biological Cybernetics, 42*, 1–8.

Barto, A.G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256.

Berliner, H.J. (1974). Chess as problem solving: The development of a tactics analyzer. Carnegie Mellon University Ph.D. dissertation (Computer Science). Pittsburgh, PA.

Berliner, H.J. (1980). Backgammon computer program beats world champion. *Artificial Intelligence* 14(1).

Berliner, H.J., & Ackley, D.H. (1982). The QBKG system: Generating explanations from a non-discrete knowledge representation. *Proceedings of the National Conference on Artificial Intelligence AAAI-82,* Pittsburgh, PA, 213–216.

Bethke, A.D. (1981). Genetic algorithms as function optimizers. University of Michigan Ph.D. Thesis, Ann Arbor, MI.

Bienenstock, E.L., Cooper, L.N., & Munro, P.W. (1982). Theory for the development of neuron activity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience, 2,* 32–48.

Booker, L.B. (1985). Improving the performance of genetic algorithms in classifier systems. In J. Grefenstette (Ed.) *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* 80–92. Carnegie Mellon University, Pittsburgh, PA.

Brady, R.M. (1985). Optimization strategies gleaned from biological evolution. *Nature, 317,* 804–806.

Bruynooghe, M. (1981). Solving combinatorial search problems by intelligent backtracking. *Information Processings Letters, 12*(1).

Burr, D.J. (1986, October). A neural network digit recognizer. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics,* 1621–1625.

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

Crutchfield, J.P., Farmer, J.D., Packard, N.H., & Shaw, R.S. (1986, December). Chaos. *Scientific American, 255*(6), 46–57.

Dechter, R. (1986). Learning while searching in constraint satisfaction problems. *Proceedings of the National Conference on Artificial Intelligence AAAI-86,* 178–183.

DeJong, K.A. (1975). Analysis of the behavior of a class of genetic algorithms. University of Michigan Ph.D. Thesis, Ann Arbor, MI.

DeKleer, J. (1983). Choices without backtracking. *Proceedings of the National Conference on Artificial Intelligence AAAI-83,* Washington, D.C., 79–85.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence, 12,* 231–272.

Dunlop, A.E., & Kernighan, B.W. pt (1985). A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design, 4*(1), 92–98.

Eldredge, N., & Gould, S.J. (1972). Punctuated equilibria: an alternative to phyletic gradualism. In T.J.M. Schopf (Ed.), *Models in Paleobiology,* 82–115. San Francisco: Freeman, Cooper and Co.

ElGamal, A., & Shperling, I. (1984, April). Design of good codes via simulated annealing. *List of Abstracts, Workshop on Statistical Physics in Engineering and Biology*, Yorktown Heights, NY.

Feldman, J.A. (1982). Dynamic connections in neural networks. *Biological Cybernetics, 36*, 193–202.

Fisher, R.A. (1930). *The Genetical Theory of Natural Selection.* Oxford: Clarendon Press.

Garey, M.R., & Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W.H. Freeman.

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

Gould, S.J. (1980). *The Panda's Thumb: More Reflections in Natural History.* New York: W.W. Norton and Co.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: Part I. Parallel development and coding of neural feature detectors. *Biological Cybernetics, 23*, 121–134.

Hebb, D.O. (1949). *The Organization of Behavior.* New York: Wiley.

Hinton, G.E. (1977). Relaxation and its role in vision. Unpublished doctoral dissertation, University of Edinburgh. Described in D.H. Ballard & C.M. Brown (Eds.), *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, 408–430.

Hinton, G.E., & Anderson, J.A. (1981). *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum.

Hinton, G.E., & Sejnowski, T.J. (1983). Optimal perceptual inference. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 448-453.

Hinton, G.E., & Sejnowski, T.J. (1986). Learning and relearning in Boltzmann Machines. Chapter 7 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

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

Holland, J.H., & Reitman, J.S. (1978). Cognitive systems based on adaptive algorithms. In D.A. Waterman & F. Hayes-Roth (Eds.) *Pattern-Directed Inference Systems*, 313–329, New York: Academic Press.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA, 79*, 2554–2558.

Horn, J. (1986, December 16). Personal communication.

Hummel, R.A., & Zucker, S.W. (1983). On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-5*, 267–287.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., & Schevon, C. (1987, in preparation). Optimization by simulated annealing: An experimental evaluation (Part I).

Jordan, M.I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 531–546. (August 15–17, 1986, Amherst, MA). Hillsdale, NJ: Lawrence Earlbaum.

Kernighan, B.W., & Lin, S. (1970). An efficient heuristic technique for partitioning graphs. *Bell Systems Technical Journal, 49*, 291–307.

Kernighan, B.W. (1986, November). Personal communication.

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

Kuhn, T.S. (1970). *The Structure of Scientific Revolutions* (2nd ed.) Chicago: The University of Chicago Press.

Kukich, K. (1987). Where do phrases come from: Some preliminary experiments in connectionist phrase generation. In G. Kempen (Ed.), *Natural Language Generation: Recent Advances in Artificial Intelligence, Psychology and Linguistics*. Dordrecht, The Netherlands: Martinus Nijhoff Publishers, a division of Kluwer Academic Press.

Le Cun, Y. (1985, June). Une procedure d'apprentissage pour reseau a seuil asymetrique [A learning procedure for asymmetric threshold network]. *Proceedings of Cognitiva 85*, 599–604, Paris.

Lenat, D.B. (1976). AM: an artificial intelligence approach to discovery in mathematics as heuristic search. Stanford University Ph.D. dissertation. Department of Computer Science report CS-570.

Lenat, D.B. (1982). The nature of heuristics. *Artificial Intelligence*.

Linsker, R. (1986a). From basic network principles to neural architecture: Emergence of spatial-opponent cells. *Proceedings of the National Academy of Sciences USA, 83*, 7508–7512.

Linsker, R. (1986b). From basic network principles to neural architectures: Emergence of orientation-selective cells. *Proceedings of the National Academy of Sciences USA, 83*, 8390–8394.

Linsker, R. (1986c). From basic network principles to neural architecture: Emergence of orientation columns. *Proceedings of the National Academy of Sciences USA, 83*, 8779–8783.

Mandelbrot, B.B. (1983). *The Fractal Geometry of Nature.* (Updated and augmented). San Francisco: W.H. Freeman.

Matwin, S., & Pietrzykowski, T. (1985). Intelligent backtracking in plan-based deduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7*(6), 682–692.

Mayr, E. (1942). *Systematics and the origin of species.* New York: Columbia University Press.

McClelland, J.L., & Rumelhart, D.E. (Eds.). (1986). *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 2: Psychological and biological models.* Cambridge, MA: The MIT Press (A Bradford Book).

Minsky, M., & Papert, S. (1969). *Perceptrons.* Cambridge, MA: MIT Press.

Minsky, M. (1987). *Society of Mind.* Simon & Schuster.

Mitchell, T.M., Carbonell, J.G., & Michalski, R.S. (Eds.). (1986). *Machine Learning: A guide to current research.* Boston: Kluwer Academic Publishers.

Newell, A., & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Newell, A., Shaw, J.C., & Simon, H.A. (1960). Report on a general problem solving program for a computer. In *Information Processing: Proceedings of the International Conference on Information Processing,* 256–264, Paris: Unesco House.

Newell, A. (1985, November 4). Personal communication.

Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4,* 135–183.

Nilsson, N.J. (1977). *Problem Solving Methods in Artificial Intelligence.* New York: McGraw-Hill.

Parker, D.B. (1985). Learning-Logic. Technical Report TR-47. Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

Rabin, M.O. (1980). Probabilistic algorithms for testing primality. *Journal of Number Theory, 12,* 128–138.

Rosenblatt, F. (1959). Two theorems of statistical separability in the perceptron. In *Mechanisation of thought processes: Proceedings of a symposium held at the National Physical Laboratory, November 1958. Vol. 1.* 421–456. London: HM Stationery Office.

Rosenblatt, F. (1961). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Washington, DC: Spartan.

Rosenblatt, F. (1962). Strategic approaches to the study of brain models. In H. von Foerster (Ed.), *Principles of Self-Organization,* Pergamon Press.

Rumelhart, D.E., Hinton, G.E., & McClelland, J.L. (1986). A general framework for parallel distributed processing. Chapter 2 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986a). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986b). Learning internal representations by error propagation. Chapter 8 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

Rumelhart, D.E., & McClelland, J.L. (Eds.). (1986a). *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.* Cambridge, MA: The MIT Press (A Bradford Book).

Rumelhart, D.E., & McClelland, J.L. (1986b). On learning the past tenses of English verbs. Chapter 18 of J.L. McClelland & D.E. Rumelhart (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 2: Psychological and biological models.*

Rumelhart, D.E., & Zipser, D. (1986). Feature discovery by competitive learning. Chapter 5 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

Samuel, A.L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3):210–229. (Reprinted in E.A. Feigenbaum & J. Feldman (Eds.) *Computers and Thought*, 71–105. New York: McGraw-Hill, 1963.)

Sejnowski, T.J., & Rosenberg, C.R. (1986). NETtalk: a parallel network that learns to read aloud. Johns Hopkins technical report JHU/EECS-86/01.

Shanks, J.L. (1969). Computation of the fast Walsh-Fourier transform. *IEEE Transactions of Computers*, 457–459.

Simon, H.A. (1981). *The Sciences of the Artificial* (2nd edition). Cambridge, MA: The MIT Press.

Skarda, C.A., & Freeman, W.J. (1987). Brains make chaos to make sense of the world. *Behavioral and Brain Sciences.*

Slate, D.J., & Atkin, L.R. (1977). CHESS 4.5 – The Northwestern University Chess Program. In P. Frey (Ed.), *Chess Skill in Man and Machine*, Springer-Verlag.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Chapter 6 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

Sutton, R.S., & Barto, A.G. (1981). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review, 88*, 135–170.

Sutton, R.S. (1984). Temporal credit assignment in reinforcement learning. University of Massachusetts Ph.D. dissertation. Department of Computer and Information Science Technical report 84-2. Amherst, MA.

Sutton, R.S. (1987). Learning to predict by the methods of temporal differences. GTE Laboratories technical report TR87–509.1 (Computer and Intelligent Systems Laboratory). Waltham, MA.

Thomas, G.B. (1968). *Calculus and analytic geometry* (4th ed.) Reading, MA: Addison-Wesley.

Vecchi, M.P., & Kirkpatrick, S. (1983). Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, CAD-2*, 215–222.

Walsh, J.L. (1923). A closed set of orthogonal functions. *American Journal of Mathematics, 55*, 5–24.

Widrow, G., & Hoff, M.E. (1960). Adaptive switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4*, 96–104.