MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Technical Report No. 1569 May, 1994

From Genetic Algorithms To Efficient Optimization

Deniz Yuret

This publication can be retrieved by anonymous ftp to publications.ai.mit.edu.

Abstract

The work described in this thesis began as an inquiry into the nature and use of optimization programs based on "genetic algorithms." That inquiry led, eventually, to three powerful heuristics that are broadly applicable in gradient-ascent programs: First, remember the locations of local maxima and restart the optimization program at a place distant from previously located local maxima. Second, adjust the size of probing steps to suit the local nature of the terrain, shrinking when probes do poorly and growing when probes do well. And third, keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid ascent.
These algorithms lie at the core of a novel optimization program that illustrates the power to be had from deploying them together. The efficacy of this program is demonstrated on several test problems selected from a variety of fields, including De Jong's famous test-problem suite, the traveling salesman problem, the problem of coordinate registration for image guided surgery, the energy minimization problem for determining the shape of organic molecules, and the problem of assessing the structure of sedimentary deposits using seismic data.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research was provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

From Genetic Algorithms
To Efficient Optimization

by

Deniz Yuret

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

The work described in this thesis began as an inquiry into the nature and use of optimization programs based on "genetic algorithms." That inquiry led, eventually, to three powerful heuristics that are broadly applicable in gradient-ascent programs: First, remember the locations of local maxima and restart the optimization program at a place distant from previously located local maxima. Second, adjust the size of probing steps to suit the local nature of the terrain, shrinking when probes do poorly and growing when probes do well. And third, keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid ascent.

These algorithms lie at the core of a novel optimization program that illustrates the power to be had from deploying them together. The efficacy of this program is demonstrated on several test problems selected from a variety of fields, including De Jong's famous test-problem suite, the traveling salesman problem, the problem of coordinate registration for image guided surgery, the energy minimization problem for determining the shape of organic molecules, and the problem of assessing the structure of sedimentary deposits using seismic data.

Thesis Supervisor: Patrick Henry Winston
Title: Director, MIT Artificial Intelligence Laboratory

Acknowledgments[1]

Many people contributed to this research by providing the various examples. I would like to thank Tomas Lozano-Pérez for the image guided surgery work, Edward Wang for the energy minimization example, Fabio Boschetti for the refraction tomography problem, Shanghua Teng for his support on the distant point algorithm, Hal Abelson for the downhill simplex implementation, Rick Lathrop for the reverse folding examples, Silvio Turrini for the work on circuit design, and others who kept this research alive with their interest. Patrick Winston is a great advisor. I must thank him for his continuous support and original ideas. This thesis would not have been written without him. Finally, Michael de la Maza deserves special thanks for his large contribution to this work and for being an excellent research partner.

[1] This report describes research done at the Artificial Intelligence laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

Contents

List of Figures

Chapter 1

Introduction

In this chapter I define the problem of optimization, explain the problems usually associated with optimization, establish criteria for success for optimization algorithms, and introduce several key heuristic ideas.

The goal of optimization is, given a system, to find the setting of its parameters that yields optimal performance. The performance measure is given as an objective function.

The cases where the parameter space can be searched exhaustively or the objective function can be subjected to analytical methods are not considered in this thesis. Thus, it is assumed that the only source of information available to an optimization algorithm is the evaluation of the objective function at a limited number of selected points as in Figure 1-1.

What makes global optimization a challenging problem are the potential existence of many local maxima, a high number of dimensions, uncooperative twists and turns in the search space, and an objective function that is costly to compute. For a general introduction to computational methods in optimization see [Polak, 1971], or [Dennis and Schnabel, 1983].

The work described in this thesis started with explorations with genetic-algorithms. Typical programs based on the genetic-algorithm idea suffer not only from all of the problems that generally plague attempts at global optimization, but also from problems that are specific to the use of populations subject to survival pressures.

In particular, local optima attract populations like magnets. The right way to maintain

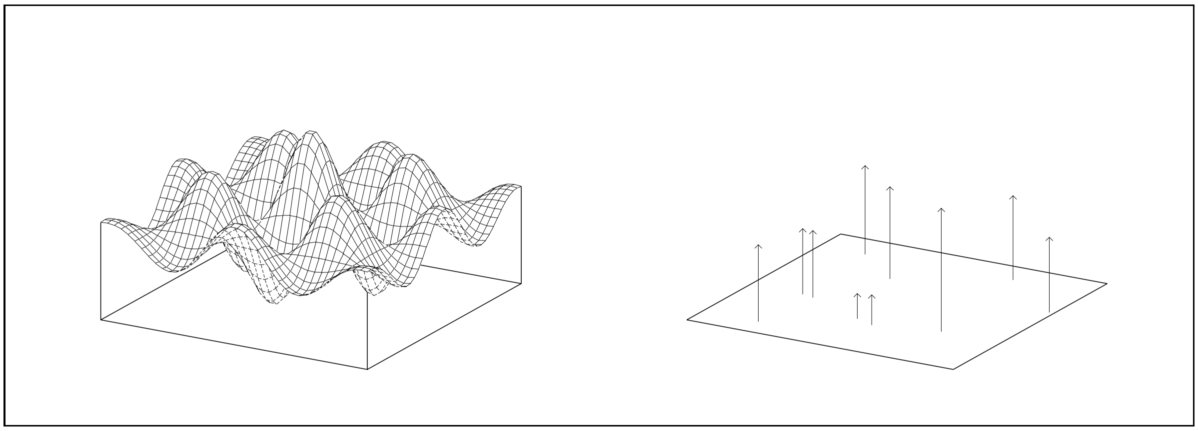


Figure 1-1: The first figure shows a typical objective function. The second shows the information available to the optimization algorithm.

diversity is not clear. Most proposed solutions end up modifying the objective function, which can result in the total loss of the global optimum [Winston, 1992], or are specific to the particular problem at hand and difficult to generalize [Hillis, 1990].

Also, the large number of individuals involved in the typical genetic program results in a large number of function evaluations. Especially in problems with a high number of parameters, this considerably slows the genetic algorithm down.

The search for a systematic approach to these problems and experiments with numerous algorithms [Yuret, 1992, Yuret and de la Maza, 1993, de la Maza and Yuret, 1994] finally led to the following three key heuristics:

- Remember the locations of local maxima and restart the optimization program at a place distant from previously located local maxima.

- Adjust the size of probing steps to suit the local nature of the terrain, shrinking when probes do poorly and growing when probes do well.

- Keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid ascent.

None of these three heuristics is new, and none of them are specific to optimization. Nevertheless, the combination leads to a powerful optimization method. The generality of the ideas ensures the algorithm's adaptability to a wide range of problems. This is justified by the empirical evidence on several real-world problems, where the method outperformed well established algorithms such as Powell's method and simulated annealing.

The next three chapters take each of these key ideas in turn, and analyze their contribution to the overall picture. This is followed by test results and analysis of related work on several problems. The last chapter concludes with a summary of the thesis and discussion of future work. Sample code is provided in the appendix.

Chapter 2

Using the local maxima

This chapter will explain the application of the first key idea as a solution to the local maxima problem:

- Remember the locations of local maxima and restart the optimization program at a place distant from previously located local maxima.

I will argue that the locations of the local maxima discovered so far can be used to guide further search in the space. The first section will introduce the problem of local maxima, and describe the initial attempts to solve it. The second section is a discussion on the optimum population size and the possible implementation on parallel computers. The third section will explain how to use the local maxima, and introduce the top level procedure for our method. The fourth section presents several solutions to the problem of finding points with high diversity. Finally the fifth section will conclude the chapter by a discussion of two other approaches inspired by natural phenomena, Hillis's idea of "co-evolving parasites", and the simulated annealing algorithm.

2.1 The problem of local maxima

Local maxima are a major problem not just for genetic algorithms, but any optimization technique that sets out to find the global optimum. A genetic algorithm works nicely in

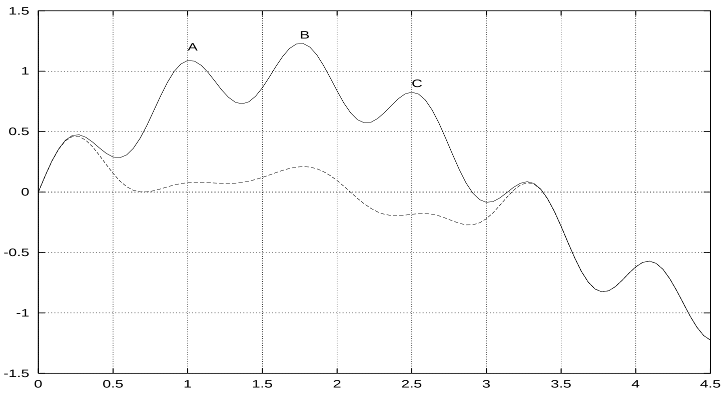


Figure 2-1: The discovery of the local maxima at A and C result in a modification of the objective function such that the global maximum at B is no longer visible.

the exploration stage, with each of the individuals discovering pieces of the solution and combining them together. However when a locally optimal point is achieved by a particular individual, it manages to hold the lead for a number of iterations and all individuals start looking alike. The leader survives through generations, and contributes in many crossovers, distributing its genes to every other candidate. The distant explorers are outperformed and gradually eliminated. Finally, progress comes to a halt when diversity ceases to exist.

The initial approach taken for this research was making diversity a part of the fitness function [Winston, 1992, Yuret, 1992]. This works by populating the local maxima in order to avoid them. When diversity is part of the fitness function, the individuals that are close to an already populated local maxima are punished, and distant explorers are rewarded. There are several difficulties with this approach. First, it is not clear how to combine fitness with diversity, and experiments [Yuret, 1992] suggest that an implementation good for one particular problem has difficulty with another. Second, as the number of discovered local maxima increases, the size of the population has to increase also. Otherwise the population "forgets" some of the local maxima and new generations keep falling into the old traps. Finally, as with any method that permanently modifies the objective function, there is the danger of covering the actual global optimum, and making it impossible to find. An example is given in Figure 2-1. An optimization algorithm should never lose the possibility of finding the global optimum.

The problem of forgetting the local maxima is particularly interesting. It suggests a dynamic population size that grows as more local maxima are discovered. However, this will reduce efficiency because simulating each generation will become more expensive. The next section presents a way to get around this problem. The section after next will illustrate how to use local maxima without modifying the evaluation function.

2.2 Parallel vs serial

> "Mutation is a sideshow in nature and a sideshow in genetic algorithms. I usually keep it turned off."
>
> John Koza, author of "Genetic Programming"

Using a large number of individuals in a genetic algorithm results in a large number of function evaluations. A small number of individuals cannot keep track of the positions of previously discovered local maxima. Thus new generations keep falling into the same traps. If a particular individual is stuck at a local maximum, then one option is freezing it, and removing it from the reproductive cycle. This way, the algorithm can keep the size of the population in the reproductive cycle, thus the number of function evaluations, manageable. This scheme was analyzed in previous work [Yuret and de la Maza, 1993]. The population is separated into two groups; young and old. The size of the young population, which contribute to reproduction, is held constant. Individuals that survive more than a maximum number of generations are moved into the old population. This makes it likely that the old are at local maxima. This way, the old population can grow and record an arbitrary number of local maxima while the number of function evaluations per generation is held constant.

Next comes the question of the ideal size for the young population. Having a large number has several advantages. But none of these advantages is enough to offset the need for a large number of function evaluations. In particular, it is claimed that different groups in the population can optimize different parts of the gene, thereby solving different subproblems. Then, the results can be combined into a single individual via crossover. First, this presumes that the representation for the gene separates different pieces of the problem into different dimensions. In other words, the evaluation function needs to be additively separable, which is typically not the case in most numerical optimization problems. Furthermore, crossover would be an advantage if these groups were searching the space in parallel. However on a serial computer, solving a separable problem via crossover takes just as much time as solving one subproblem first, and then solving the other. Note that no crossover is necessary in the serial scheme. A favorable trait is immediately spread to all the individuals, so the individual that solves the next subproblem will actually include the solution to both subproblems.

In this thesis, I will present an algorithm that has only one active individual. Experiments suggest that, with an intelligent mutation scheme, this method is more efficient than using multiple individuals. The local maxima will be kept on a separate list, and the knowledge of their location will be used to guide the search. On a parallel computer, every processor can keep track of a separate individual searching a different region in space. The information they provide on the locations of local maxima can be combined and used to redirect individuals who get stuck. Thus the algorithm naturally accepts a parallel implementation. One might choose to focus the search into regions far away from previously discovered local maxima, or concentrate it in a region where the local maxima are most dense. In the next section I will explore the former strategy. However, there can be no one single strategy that will work well on all problems. A combination of different strategies can be the most effective solution.

2.3 How to use the local maxima

There is no way to avoid local maxima (after all, among them we may find the global one we are looking for). Thus the optimization algorithm should make use of the information gained by the ones discovered. The main objective of adding diversity to the evaluation function, or modifying it, is to mark the discovered local maxima and avoid them. Doing this directly, without modifications to the objective function, gives better results.

There are several different ways to use the positions of local maxima. The algorithm can keep them for use in future cross-overs. This will give good results in problems where combining several good solutions leads to a better solution. However as the number of local

OPTIMIZE(f)
1 $X \leftarrow \{\vec{x_{min}}, \vec{x_{max}}\}$
2 $\vec{x} \leftarrow$ DISTANT-POINT(X)
3 $X \leftarrow X \cup$ LOCAL-OPTIMIZE($f, \vec{x}$)
4 **goto** 2

Procedure 2-1: The top level loop of the optimization. f is the objective function, $\vec{x}$ is the starting point, X is the set of discovered local maxima. It is initialized to contain the borders of the search space.

maxima increases, this results in a high number of function evaluations. Alternatively, the algorithm can just use the locations of local maxima to avoid getting trapped in them again. A simple approach would be to take a random step when stuck at a local maximum, much like simulated annealing. Then one can hope that the search will evolve into a different region of the space instead of converging back to the same point. This has the advantage of staying around good solutions. However, it is not clear how far to jump when stuck. This approach also has the disadvantage of ignoring all but the last one of the local maxima. The method I will describe in this section uses the locations of all local maxima in order to keep away from them. This can be achieved by recording the local maxima as they are discovered, and directing the search to unexplored regions of the space. This suggests Procedure 2-1.

LOCAL-OPTIMIZE can be any popular optimization procedure that will converge to local

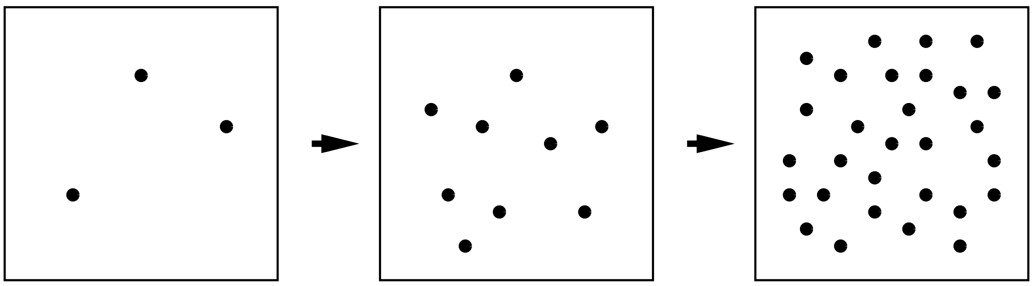


Figure 2-2: The idea of iterative deepening: The algorithm has a reasonable solution whenever the user wants to terminate. When there is more time available, it keeps exploring the space in ever increasing detail.

maxima. One such procedure is the subject of the next few chapters. DISTANT-POINT selects a new point in an unexplored region of the space using the local maxima discovered so far that are stored in X. How this can be done efficiently is the topic of the next section.

Note that this algorithm does not modify the evaluation function in any way. It just restarts the search in unexplored regions of the space. Figure 2-2 shows how this results in a uniform exploration of the search space.

There is one more little mystery in the algorithm. The loop between lines 2 and 4 never terminates. This is reminiscent of the idea of iterative deepening in computer chess literature. Contemporary chess programs are designed in such a way that they keep searching until the time is up. Thus, they always have to have a reasonable move ready from what they have discovered so far. Global optimization, like chess, is a problem of searching a very large space. Thus one can never be sure of the answer until the whole space is exhausted. This is practically impossible in many cases. A good search algorithm should save the reasonable solutions it has found, and keep searching for better ones as time permits.

2.4 How to find distant points

In the previous section I basically stated the objective of the procedure DISTANT-POINT. More precisely, given a number of points in a bounded space, DISTANT-POINT should find a new point that has the maximum distance to its nearest neighbor. The known algorithms tend to explode in high dimensional spaces. One of the motives behind the method proposed in this thesis is being scalable to problems with high number of dimensions. Thus a simple algorithm which has linear cost is proposed. Another approach is to choose a few random points and pick the best one in diversity as the new starting point. It is shown that in very high number of dimensions, the distance of a random point to its nearest neighbor is very likely to be close to the maximum.

The problem is related to the nearest neighbor problem [Teng, 1994]. Thus it is no surprise that Voronoi diagrams which are generally used in finding nearest neighbors are also the key in solving the distant point problem. Figure 2-3 illustrates the basic principle in two dimensions.

A Voronoi diagram consists of convex regions that enclose every given point. Each convex region is the set of points that are closest to the given point in that region. Thus the distant point has to lie on one of the borders. More specifically it will be on one of the corners of the convex Voronoi regions. So a procedure that gives provably correct results, is to check all corners exhaustively, and report the one that maximizes distance to its nearest neighbor. However the number of corners in a Voronoi diagram increase very rapidly with the number of dimensions. Thus this procedure is impractical if the number of dimensions exceed a few.

To cope with high dimensional spaces a more efficient approximation algorithm is necessary. Figure 2-4 demonstrates a solution in one dimension. Each time a new point is required, the algorithm finds the largest interval between the existing points. It then returns a random point within that interval. A simple extension of this to high number of dimensions is to repeat the same process independently for each dimension and construct the new point from the values obtained. Although this scheme does not give the optimum answer, it is observed to produce solutions that are sufficiently distant from remembered local maxima. Procedure 2-2 is the pseudo-code for this algorithm.

Another option is to choose the new points randomly. Analysis shows that in a very high dimensional space, the distance of a randomly selected point to its nearest neighbor will be

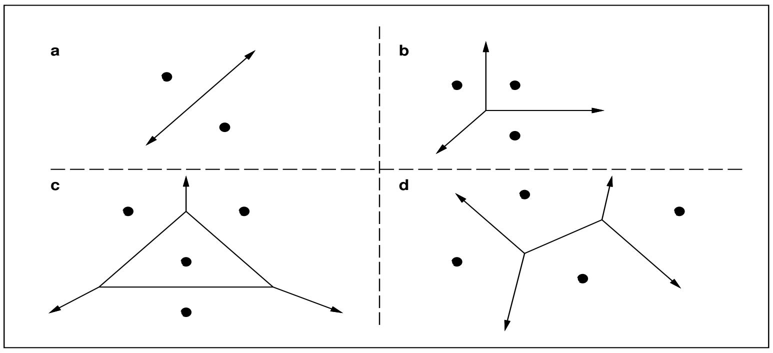


Figure 2-3: Voronoi diagrams: Each region consists of points that are closest to the point inside that region.

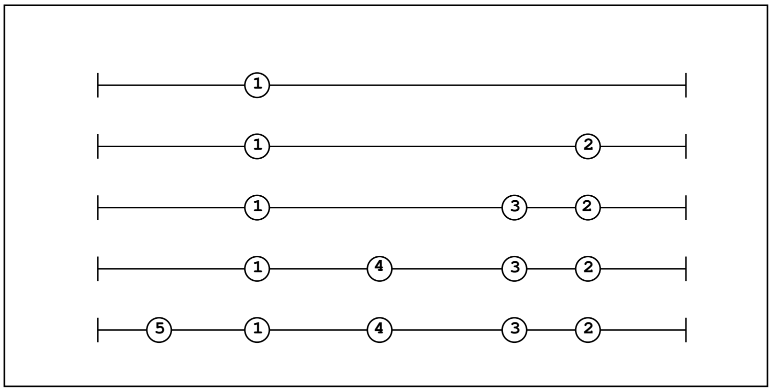


Figure 2-4: Execution of the distant point algorithm in one dimension. First the largest interval between the existing points is found, then a random point within that interval is returned.

DISTANT-POINT(X)
$\vec{x} \leftarrow \vec{0}$
for each dimension d
  SORT $\vec{x_i} \in$ X s.t. $\forall i\ \vec{x_{i-1}}[d] \leq \vec{x_i}[d]$
  FIND $\vec{x_j} \in$ X s.t. $\forall i\ (\vec{x_j}[d] - \vec{x_{j-1}}[d]) \geq (\vec{x_i}[d] - \vec{x_{i-1}}[d])$
  $\vec{x}[d] \leftarrow$ RANDOM$(\vec{x_{j-1}}[d], \vec{x_j}[d])$
return $\vec{x}$

Procedure 2-2: The distant point algorithm. $\vec{x}[d]$ is used to represent the d'th component of the vector $\vec{x}$. RANDOM(x,y) returns a random value between x and y. Every component of the new vector is set to a random value within the largest empty interval in that dimension.

close to maximum with very high probability. To simplify the argument, let us assume that the search space has the shape of a sphere, and that we have found one local maximum at the center. The question is, how far away a chosen random point will fall. Figure 2-5 compares the radii of spheres in different number of dimensions with their half sized spheres. Note how the border of the half sized sphere approaches the large one as the number of dimensions grow. The probability of a random point falling into a certain region is proportional to the volume of that region. For example, if a random point is chosen in a ten dimensional space, there is a 0.5 probability of it falling into the narrow strip between the two spheres. Thus, with high probability, it will be away from the center, and close to the border. We can extend this argument to spaces of different shapes with more local maxima in them. The important conclusion is that, if a problem has many dimensions, it is not worthwhile to spend much effort to find the optimal distant point. A point randomly chosen is likely to be sufficiently distant.

In the results I present in this thesis, I preferred to use the DISTANT-POINT algorithm presented in Procedure 2-2 for low dimensional problems. For problems with a high number of dimensions (e.g. the molecule energy minimization problem with 153 dimensions), I chose random points instead.

2.5 How does nature do it

There are various other approaches to the problem of local maxima inspired by natural phenomena. In this section Hillis' *co-evolving parasites* [Hillis, 1990] inspired by biology, and the *simulated annealing algorithm* [Kirkpatrick et al., 1983] inspired by physics, will be discussed.

Hillis used simulated evolution to study the problem of finding optimum sorting networks. To tackle the local maxima problem he proposed to use *co-evolving parasites*. In his implementation, the evolution is not limited to the sorting networks. The sorting problems also evolve to become more difficult. This is possibly how nature manages to keep its di-

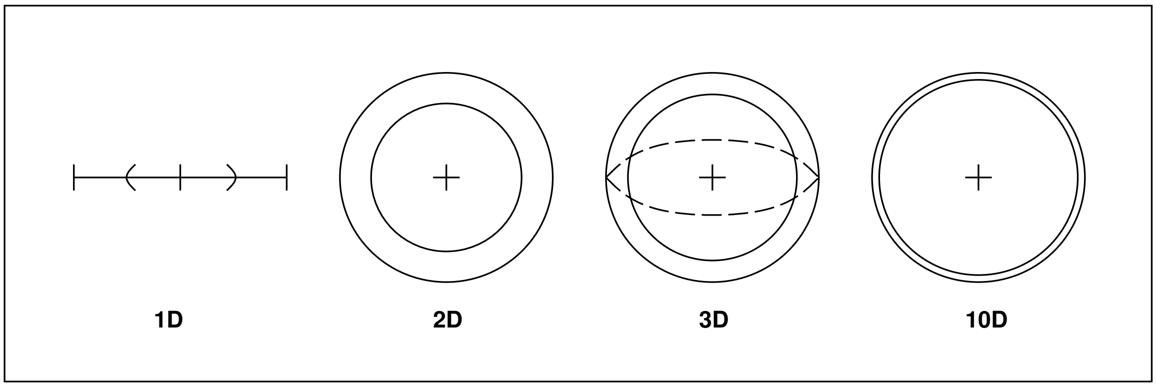


Figure 2-5: The comparison of spheres and half spheres in different number of dimensions

versity. The problem facing each species is never constant, the environment keeps changing, and more importantly the rival species keep adapting. When the predator develops a better weapon, the prey evolves better defense mechanisms. As a result nature has an adaptive sequence of problems and solutions. The domain Hillis chose is particularly suitable for this approach, because it provides many problem instances. However, his method is not applicable in general. For example, if we want to optimize the parameters for the design of an aircraft, there is nothing corresponding to the multiple problem instances to evolve.

Simulated annealing is one of the popular methods for optimization in spaces with a lot of local maxima. It allows non-optimal moves in a probabilistic fashion. Many other algorithms for optimization belong to the class of greedy algorithms [Cormen et al., 1990]. The algorithms in this class always make the choice that looks best at the moment. They hold on to the best point found so far and never step down when climbing a hill. This is exactly the reason why greedy algorithms get stuck at local maxima. Simulated annealing, on the other hand, by not taking the greedy approach, is able to jump out of local maxima. However, this sacrifices efficiency. Probabilistically allowing non-optimal moves does not seem like the best use of information gained from the search.

There is another interesting aspect to simulated annealing. It uses a schedule of temperature which determines the gradual change in the probability. Optimization starts at a very "hot" state, in which there is a high probability of making non-optimal moves. It slowly "cools down" as the optimization proceeds, turning itself into a hill climbing algorithm in the limit. This results in a gradual shift from a *coarse-grained* search in the beginning towards a *fine-grained* search in the end. The idea of moving from a coarse-grained to a fine-grained search is the key to the following chapter. However, this progression is achieved by decreasing the size of the steps taken in the search space rather than allowing for steps that may take the evaluation function down like simulated annealing.

Chapter 3

Changing the scale

This chapter will explain the application of the second key idea as the backbone of the local optimization algorithm:

- Adjust the size of probing steps to suit the local nature of the terrain, shrinking when probes do poorly and growing when probes do well.

In the previous chapter we concluded by introducing the idea of moving from a coarse-grained search to a fine-grained search. In this chapter I will demonstrate how this can be achieved by using a schedule of step sizes that start with large moves, then gradually shrink down to smaller ones. Instead of using a pre-determined schedule, the algorithm will adapt its own step size to the landscape according to the following principle: *Expand when making progress, shrink when stuck.* We will see how this contributes to both the efficiency and the robustness of the optimization. The next chapter will illustrate how we can further improve the efficiency by adding a small memory of recent moves.

3.1 The local optimizer

Procedure 3-1 is the pseudo-code for the primitive version of LOCAL-OPTIMIZE. The procedure takes steps in random directions. If a step leads to a better point, the step size is doubled. If the step fails, another one is tried. If too many steps fail in a row, the step size

```
LOCAL-OPTIMIZE(f, x⃗)
initialize v⃗
while |v⃗| ≥ THRESHOLD
    iter ← 0
    while f(x⃗ + v⃗) < f(x⃗) and iter < MAXITER
            v⃗ ← RANDOM-VECTOR(v⃗)
            iter ← iter + 1
    if f(x⃗ + v⃗) < f(x⃗)
            v⃗ ← v⃗/2
    else
            x⃗ ← x⃗ + v⃗
            v⃗ ← 2v⃗
return x⃗
```

Procedure 3-1: The primitive local optimize algorithm. $\vec{v}$ is the step vector. Its size grows or shrinks according to the recent progress made by the algorithm. RANDOM-VECTOR($\vec{v}$) is just a procedure that returns a random vector with the same size as $\vec{v}$. MAXITER determines how many iterations at each step size before shrinking the vector. THRESHOLD is the smallest step size allowed.

is reduced to half. The best point is returned, when the step size shrinks below a certain threshold.

With a typical genetic algorithm, it is not easy to tell whether the algorithm has located an exact maximum. Procedure 2-1 is based on staying away from the local maxima. If LOCAL-OPTIMIZE is not able to give an exact location of a maximum, the search will be pushed away from that region, and it will be a long time before the algorithm finds the better points nearby. Thus LOCAL-OPTIMIZE, by ending with a fine grained search with small step sizes, aims at locating the maxima precisely.

It is advisable to shrink and expand rapidly when using large steps, and try out more random directions when about to go below the THRESHOLD. This can be achieved by making MAXITER a function of the step size. The algorithm should try more random vectors at smaller step sizes.

For high dimensional problems, choosing random vectors can be costly and not useful. In

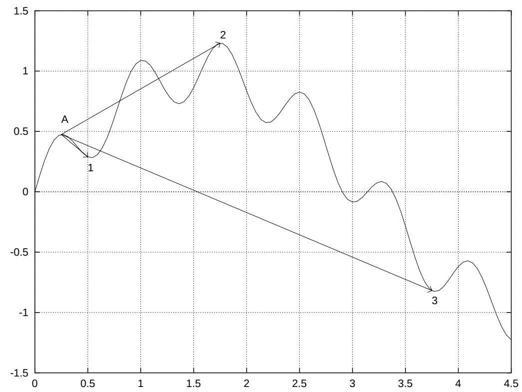


Figure 3-1: Starting with large steps is helpful for ignoring small local maxima at the initial stage of the search.

that case the RANDOM-VECTOR procedure can return vectors in unit directions. This is the convention I adopted for the results chapter. More precisely, a vector in a unit direction and its opposite were tried at all step sizes except for the smallest step size, where the algorithm cycled through all the unit directions.

3.2 Why change the step size

Changing the step size according to recent performance adapts the scale of the search to the local terrain in the search space. It does not only move from coarse grained search to fine grained search, but it is able to re-expand if the function becomes flatter. There are many situations in which, adapting step sizes helps the algorithm be more efficient and robust. First, by starting with large step sizes, the algorithm is able to ignore the small local maxima at the initial stage of the search. Secondly, in simple functions, changing the step size in multiples of 2, leads to logarithmic behavior in the distance to the maximum. This is standard in typical *line-search* techniques, however my algorithm does not restrict itself to a line and adapts its direction – as will be explained in the next chapter – simultaneously with the step size. By not diving into detailed line minimizations, the algorithm is able to locate promising regions faster. Finally, a smaller step size may help determine the right direction. Once a good direction is found, the algorithm can move faster by expanding again.

Figure 3-1 illustrates the advantage of adapting step sizes in the presence of many local maxima. Taking different sized steps from A, the algorithm can discover that the function is decreasing in the immediate neighborhood (1), it is increasing in the medium scale (2), or that the general direction is down (3). Starting with a coarse scale lets the algorithm ignore the low level detail of the search space and discover general regions that are promising. By ignoring the small peaks, the algorithm in effect performs gradient ascent on a much smoothed function. As the scale becomes finer grained, small step sizes give the exact location of the peaks.

The number of steps it takes a hill climber to traverse a path, is linear in the length of the path, if it is using constant step sizes. In contrast, LOCAL-OPTIMIZE, by changing the size of its steps in multiples of 2, requires logarithmic time. For problems where the solution consists of traveling a certain distance in each dimension, this gives us a simple running time of:

$\Theta(\#dimensions \times log_2(distance))$

However, this is not the main advantage. There are several algorithms in the literature [Dennis and Schnabel, 1983, Polak, 1971] that do better under certain assumptions about their objective function. Typical numerical optimization algorithms perform a sequence of line optimizations when faced with high dimensional problems. The advantage of this algorithm is the fact that it *does not* perform line optimizations. The best value of a certain parameter might change with the settings of the other parameters. In the genetic algorithms literature, this situation is referred to as "epistasis", which means the suppression of the effect of a gene by a nonallelic gene. Mathematically, epistasis corresponds to the class of fitness functions that are not additively separable. Such functions are common in optimization problems. Thus, an early convergence of the optimization in order to find the exact maximum on a line may prove useless. Performing a full line-search contradicts with the idea of moving from coarse grained search to fine grained search, and starts fine tuning before the rough picture is ready.

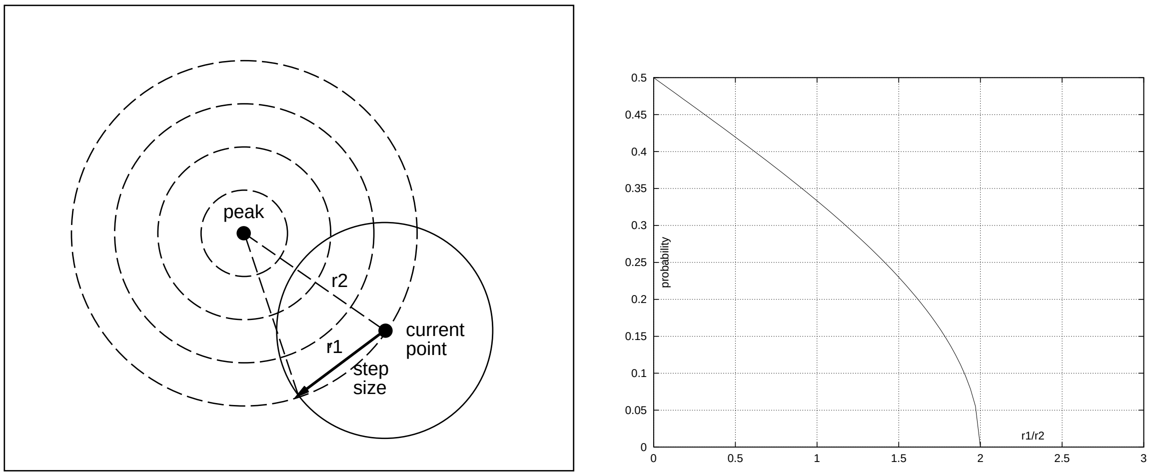


Figure 3-2: The probability of success of a random step in two dimensions as a function of the step size and the distance to the peak. Dashed circles are contour lines, the solid circle is the potential points the step can take.

3.3 How to discover the right direction

Another common problem of complex search spaces is the existence of ridges. A ridge can be defined as a region of the search space where only a small fraction of the possible directions lead to a better value. How can a hill climber, with its limited number of steps, be sure that it has arrived at a peak and is not stuck on top of a ridge? This problem is common to any search algorithm that uses a fixed set of directions. This section will demonstrate how adapting step sizes provide an answer to this problem.

Figure 3-2 illustrates the basic principle in two dimensions. The concentric circles to the left of the figure are the contour map of the objective function. It is a two-dimensional, unimodal function. The maximum is at the center. The solid circle represents the step size. The graph on the right shows the probability that a random step will succeed in moving upward as a function of the ratio of r1, which is the size of the step, to r2, the distance to the peak.

The important observation here is that the probability approaches 0.5 as the step size goes to 0. Intuitively, as the radius gets smaller, the iso-curve starts looking like a straight line (or a flat surface in higher dimensions) that cut our circle exactly in half. This is a

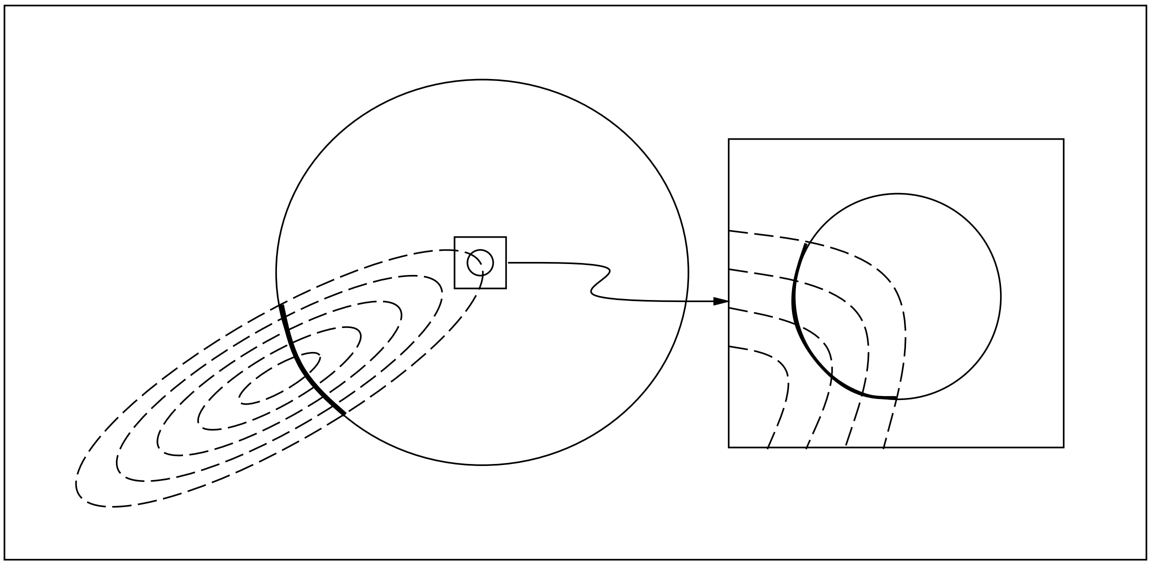


Figure 3-3: How shrinking helps on ridges. The heavy arcs show the points that are better than the current point. When the radius is smaller, as shown in the blow-up on the right, this arc gets larger.

general result that holds in any number of dimensions:

- When the step size gets smaller, the probability of a successful step approaches 0.5.

This assumes some smoothness in the objective function. Thus the choice of representation becomes important. There is a vast amount of work in genetic algorithms literature, on how to choose the right representations for certain problems. Every optimization algorithm has its ideal representation for a problem. The standard genetic algorithm favors the representations that make the problem separable, i.e. different parts of the solution can be discovered independently, and combined together. The method presented here, requires a representation in which one is able to use steps of various sizes. The larger step sizes should be more likely to bring about larger changes, and the small ones should locate the exact location of maxima. In most numerical problems this is natural. In combinatorial problems, one has to have a way to measure step size, even though the space is not Euclidean. I will present an example on the traveling salesman problem in the results chapter.

Figure 3-3 shows how adapting step sizes can help the algorithm not get stuck near ridges. This time the contours have the shape of narrow ellipses. Thus a large step size has a low probability of success. However after failing several times, the Local-Optimize starts to shrink its step size. The blow-up on the right shows how the picture looks when the step size gets smaller. The sharp edges of the contours look flatter, so the arc of points with better values than the current point, approaches a half circle.

Although shrinking down saves the algorithm from getting stuck, it also makes it slower. In search spaces where the direction of the ridge is changing, the local optimizer has to move with a lot of small zig-zags. However, augmenting it by adding by adding a small memory of recent moves that were successful, the performance increases dramatically. This is the topic of the next chapter.

Chapter 4

Using the successful moves

This chapter will explain the application of the third key idea as a means of improving efficiency:

- Keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid ascent.

Last chapter, we introduced the difficulty of traversing narrow ridges in the search space. In objective functions for which the gradient information is not available, finding a favorable direction to move is a hard problem. I propose an adaptive solution, where the algorithm uses its best guess for the gradient direction, and updates this guess after successful moves. This approach improves efficiency substantially—sometimes by an order of magnitude—and is scalable to high dimensional problems.

4.1 The ridge problem

Figure 4-1 gives a closer look at the ridge problem. Hill climbing algorithms typically take orthogonal steps, e.g. they might move in one dimension at a time. The objective function in the figure has a long, narrow ridge with a tilted axis of symmetry. The optimum direction of change has components in both dimensions. The arrows show the behavior of a procedure restricted to use two orthogonal vectors. Unless the ridge is optimally oriented, this procedure

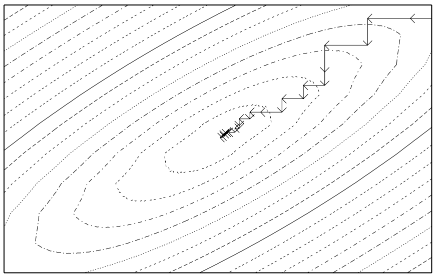


Figure 4-1: The ridge problem: The concentric ellipses are the contour map of an objective function. A hill-climbing algorithm restricted to moves in a single dimension has to go through many zig-zags to reach the optimum.

is very inefficient, taking many small steps to get to the maximum. The local optimizer presented in the previous chapter suffers from this problem. Although it is not restricted to orthogonal directions and chooses its steps randomly, without a memory of previous steps, it is unable to keep the right direction.

Figure 4-2 illustrates the problem with trying to use systematic combinations of the main directions. In one dimension we only have a +1 direction and a -1 direction. In two dimensions, in addition to the four main directions, we have 4 more secondary directions that are combinations of the main ones. In three dimensions the total number goes up to 26. The step has n components in n dimensions, and each of them can take one of three values, {-1,1,0}. We subtract away the origin and we have $3^n - 1$ directions in n dimensions.

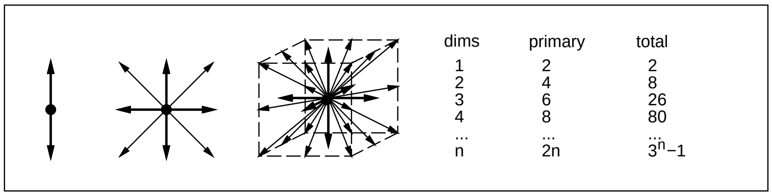


Figure 4-2: The number of combinations of main directions increase exponentially with the number of dimensions.

4.2 How to walk on ridges

Real evolution finds the right direction by trying and eliminating many wrong ones. It is not known to have a memory of the previous changes that were successful. Instead, many individuals are generated that have particular characteristics in different dimensions. The individuals that have moved in the wrong direction are simply eliminated. There is an excessive number of individuals at nature's disposal, and there is nothing wrong with wasting a few. However, for an optimization algorithm of practical value, every time step is valuable. Instead of wasting resources by using a large population, I propose gaining efficiency with a short term memory.

Note that in Figure 4-1, the right direction to move could be discovered by summing a few successful steps. This is exactly the approach that will be taken.

Adding up too many vectors, however, is dangerous. In Figure 4-1, things would have worked fine, because there is one right direction to go, but for a function like Rosenbrock's saddle in Figure 4-3, the right direction is constantly changing. Experiments with high dimensional optimization problems suggest that, the turning ridge problem is a common one. Thus, the algorithm should be able to adapt itself to the local terrain it is moving through.

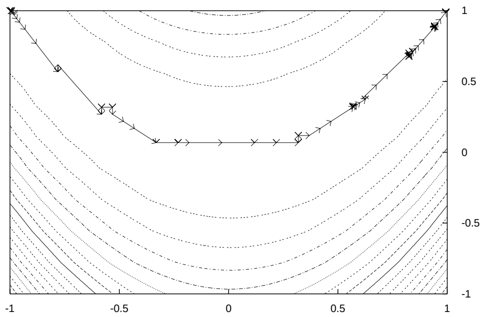


Figure 4-3: Rosenbrock's saddle. The function has a ridge along $y = x^2$ that rises slowly as it is traversed from (-1, 1) to (1, 1). The arrows show the steps taken by the improved LOCAL-OPTIMIZE. Note how the steps shrink in order to discover a new direction, and grow to speed up once the direction is established.

LOCAL-OPTIMIZE($f, \vec{x}$)

```
initialize v⃗;  u⃗ ← 0⃗
while |v⃗| ≥ THRESHOLD
    iter ← 0
    while f(x⃗ + v⃗) < f(x⃗) and iter < MAXITER
        v⃗ ← RANDOM-VECTOR(v⃗)
        iter ← iter + 1
    if f(x⃗ + v⃗) < f(x⃗)
        v⃗ ← v⃗/2
    else if iter = 0
        x⃗ ← x⃗ + v⃗;  u⃗ ← u⃗ + v⃗;  v⃗ ← 2v⃗
    else if f(x⃗ + u⃗ + v⃗) > f(x⃗)
        x⃗ ← x⃗ + u⃗ + v⃗;  u⃗ ← u⃗ + v⃗;  v⃗ ← 2u⃗
    else
        x⃗ ← x⃗ + v⃗;  u⃗ ← v⃗;  v⃗ ← 2v⃗
return x⃗
```

Procedure 4-1: The local optimize algorithm with short term memory. $\vec{v}$ is the step vector, $\vec{u}$ is the accumulator of the moves made in the same direction. Figure 4-4 illustrates how the two vectors are updated.

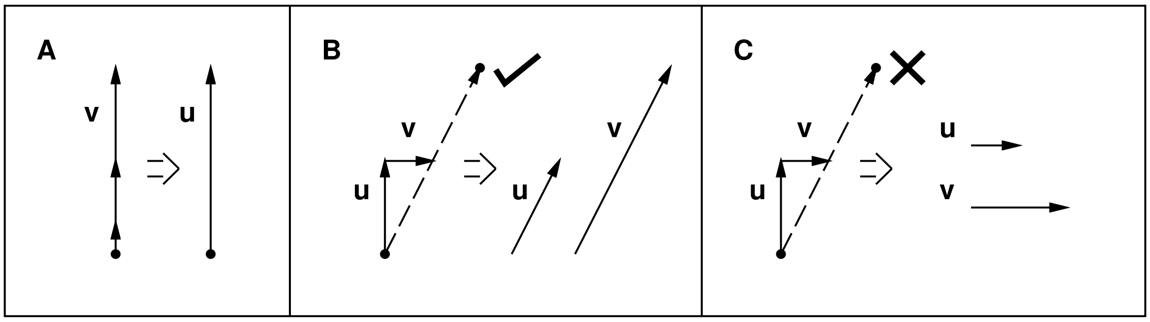


Figure 4-4: An illustration of how Procedure 4-1 uses the vectors $\vec{u}$ and $\vec{v}$. In (A) the step vector $\vec{v}$ keeps growing in the same direction as long as better points are found. $\vec{u}$ accumulates the moves made in this direction. Once $\vec{v}$ fails, the algorithm tries new directions. In (B) a new direction has been found. The combination $\vec{u} + \vec{v}$ is tried and found to give a better point. In this case $\vec{u}$ starts accumulating moves in this new direction. In (C) $\vec{u} + \vec{v}$ did not give a better point. Thus we start moving in the last direction we know that worked, in this case $\vec{v}$.

The LOCAL-OPTIMIZE in Procedure 3-1 of the previous section, is able to adapt its step sizes to the local terrain. Procedure 4-1 gives an improved version which is able to adapt its direction also. The local optimizer moves along its best estimate of the gradient direction. Vector $\vec{u}$ is used to accumulate moves made along this direction. When this direction fails, vector $\vec{v}$ enters its random probing mode, and searches until it can find a new direction that works. When a new direction is found, the algorithm tries $\vec{u} + \vec{v}$ as a new estimate of the gradient direction. Either this succeeds, and $\vec{u}$ starts accumulating moves along $\vec{u} + \vec{v}$, or it fails and the new $\vec{v}$ becomes the new estimate. Figure 4-4 explains the algorithm in pictures. As shown in the figure, the algorithm keeps adapting its estimate of the gradient vector according to the local terrain.

The performance of the algorithm improved by an order of magnitude better in complex spaces. Figure 4-3 is the result of running this algorithm on Rosenbrock's saddle. The function has a ridge along the $y = x^2$ parabola that gradually rises from (-1,1) to (1,1). The global maximum is at (1,1) and the algorithm was started on the other end of the ridge, so that it had to traverse the whole parabola. As illustrated in Figure 4-5 the more adaptive Procedure 4-1 is more than an order of magnitude efficient than Procedure 3-1 of the last chapter.

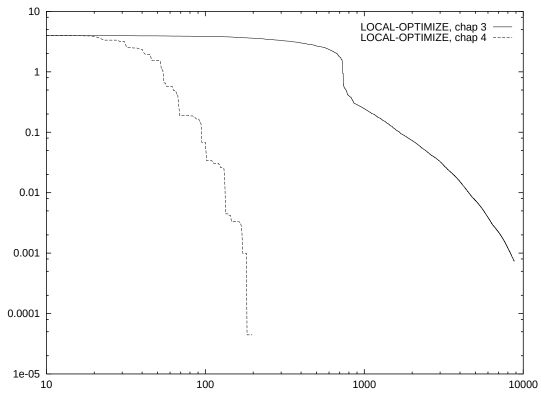


Figure 4-5: Comparison of the two versions of LOCAL-OPTIMIZE from chapter 3 and chapter 4 on Rosenbrock's saddle. Note that both axes are logarithmic. The old version converges in 8782 iterations, the new version finds a better value in 197 iterations.

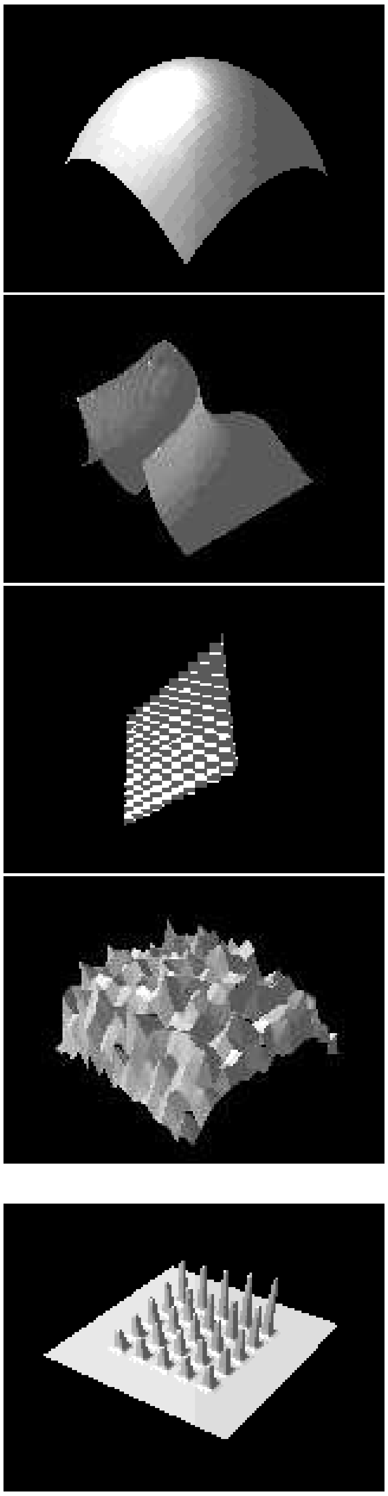
Chapter 5

Results and related work

5.1 De Jong's test suite

In this section I will present the performance of various optimization algorithms on De Jong's five function test suite. De Jong first suggested this set in his work "An analysis of the behaviour of a class of genetic adaptive systems" [De Jong, 1975]. He chose these functions because they represent the common difficulties in optimization problems in an isolated manner. By running the comparisons on these functions, one can make judgments about the strengths and weaknesses of particular algorithms. Also, De Jong's functions are quite popular in genetic algorithms literature, so it is possible to make direct comparisons.

I should emphasize that all these results are only obtained from particular implementations of these algorithms. In particular the GENEsYs package [Bäck and Hoffmeister, 1992] was used for the genetic algorithms, the ASA package [Ingber, 1993] was used for simulated annealing, and the programs from "Numerical Recipes in C" [Press et al., 1992] were used for implementations of Powell's method and downhill simplex. Other implementations, or other parameter settings may give different results. Furthermore, the topic of optimization has a vast literature. I discovered that the M.I.T. library has 1240 volumes on the subject, when I tried to do a literature search. Thus it was impossible to run comparison tests with all known methods. Instead I compared my program to a subset of well known, widely used



SPHERE MODEL:

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$$
$$-5.12 \leq x_i \leq 5.12$$
$$min(f_1) = f_1(0, \ldots, 0) = 0$$

GENERALIZED ROSENBROCK'S FUNCTION:

$$f_2(\vec{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$
$$-5.12 \leq x_i \leq 5.12$$
$$min(f_2) = f_2(1, \ldots, 1) = 0$$

STEP FUNCTION:

$$f_3(\vec{x}) = 6 \cdot n + \sum_{i=1}^{n} \lfloor x_i \rfloor$$
$$-5.12 \leq x_i \leq 5.12$$
$$min(f_3) = f_3([-5.12, -5), \ldots, [-5.12, -5)) = 0$$

QUARTIC FUNCTION WITH NOISE:

$$f_4(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + gauss(0, 1)$$
$$-1.28 \leq x_i \leq 1.28$$
$$min(f_4) = f_4(0, \ldots, 0) = 0$$

SHEKEL'S FOXHOLES:

$$\frac{1}{f_5(\vec{x})} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$
$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \cdots & 32 & 32 & 32 \end{pmatrix}$$
$$K = 500 \quad ; \quad f_5(a_{1j}, a_{2j}) \approx c_j = j$$
$$-65.536 \leq x_i \leq 65.536$$
$$min(f_5) = f_5(-32, -32) \approx 1$$

Figure 5-1: De Jong's functions. The graphs are rotated and modified by order preserving transforms for clarity.

methods.

Figure 5-1 shows the graphs and equations for the five De Jong functions. Below are their brief descriptions, and the problems they represent:

- SPHERE is the dream of every optimization algorithm. It is smooth, it is unimodal, it is symmetric and it does not have any of the problems we have discussed so far. The performance on SPHERE is a measure of the general efficiency of the algorithm.

- ROSENBROCK is the nightmare. It has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Algorithms that are not able to discover good directions underperform in this problem.

- STEP is the representative of the problem of flat surfaces. Flat surfaces are obstacles for optimization algorithms, because they do not give any information as to which direction is favorable. Unless an algorithm has variable step sizes, it can get stuck on one of the flat plateaus.

- QUARTIC is a simple unimodal function padded with noise. The gaussian noise makes sure that the algorithm never gets the same value on the same point. Algorithms that do not do well on this test function will do poorly on noisy data.

- FOXHOLES is an example of many (in this case 25) local optima. Many standard optimization algorithms get stuck in the first peak they find.

Figure 5-2 gives comparative results of my program to four well known algorithms, the standard genetic algorithm [Holland, 1975], simulated annealing [Kirkpatrick et al., 1983], Powell's method [Powell, 1964], and downhill simplex [Nelder and Mead, 1965].

Note that the standard genetic algorithm is the least efficient of all. It usually does the most number of function evaluations. The numerical algorithms, Powell's method and downhill simplex, do well in terms of efficiency. However they are not robust in the existence of flat surfaces (step function), or local optima (shekel's foxholes). Adaptive simulated annealing is able to solve all the problems consistently, although it is not as efficient as the numerical algorithms. My program, not only finds the global optimum in all five cases, its efficiency is comparable, if not better, than the other ones.

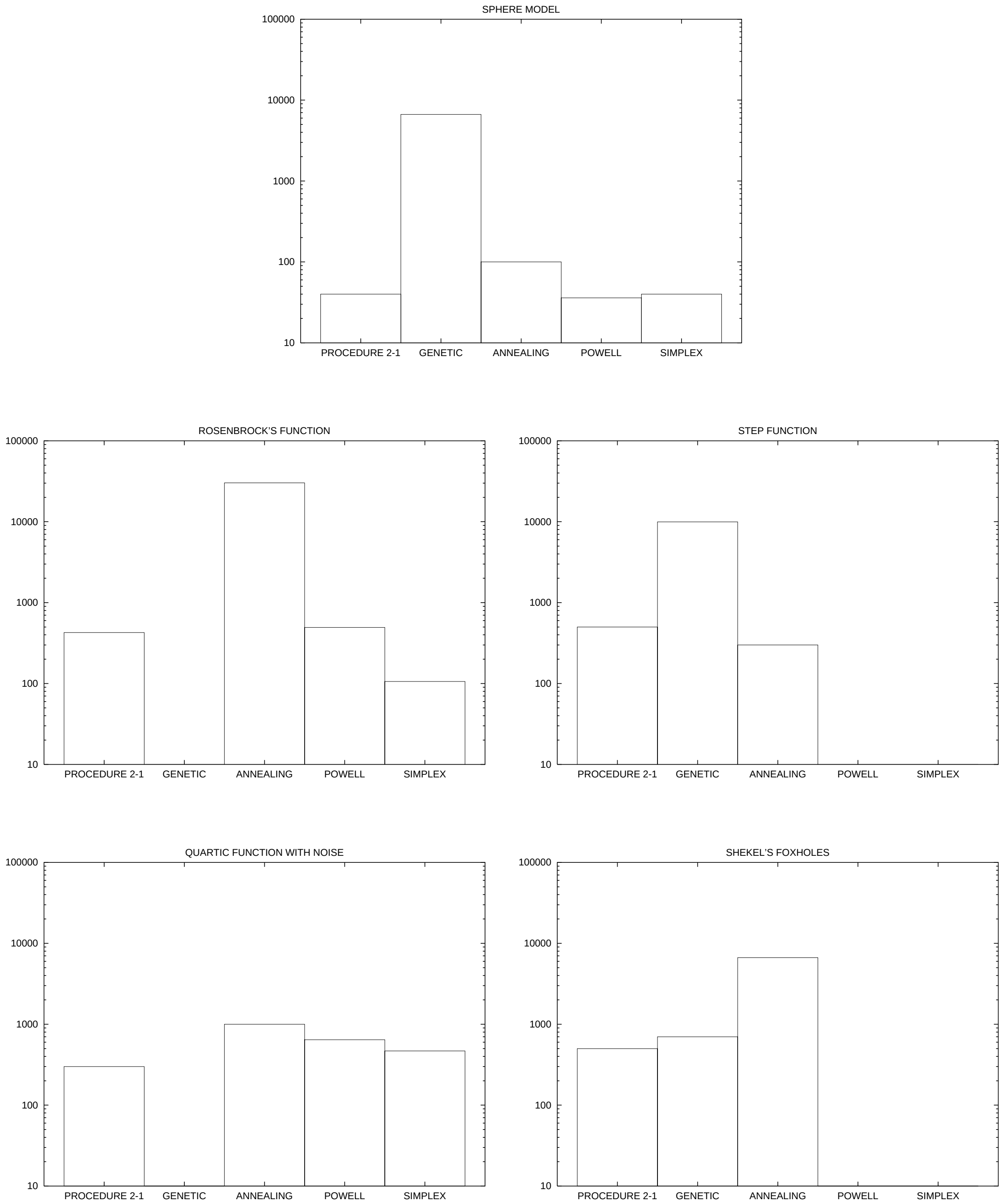


Figure 5-2: Comparison of five algorithms on De Jong's functions. The vertical bars represent the number of function evaluations to find the global optimum. The scale is logarithmic and the numbers are averaged over ten runs. No bar was drawn when an algorithm could not find the global optimum consistently.

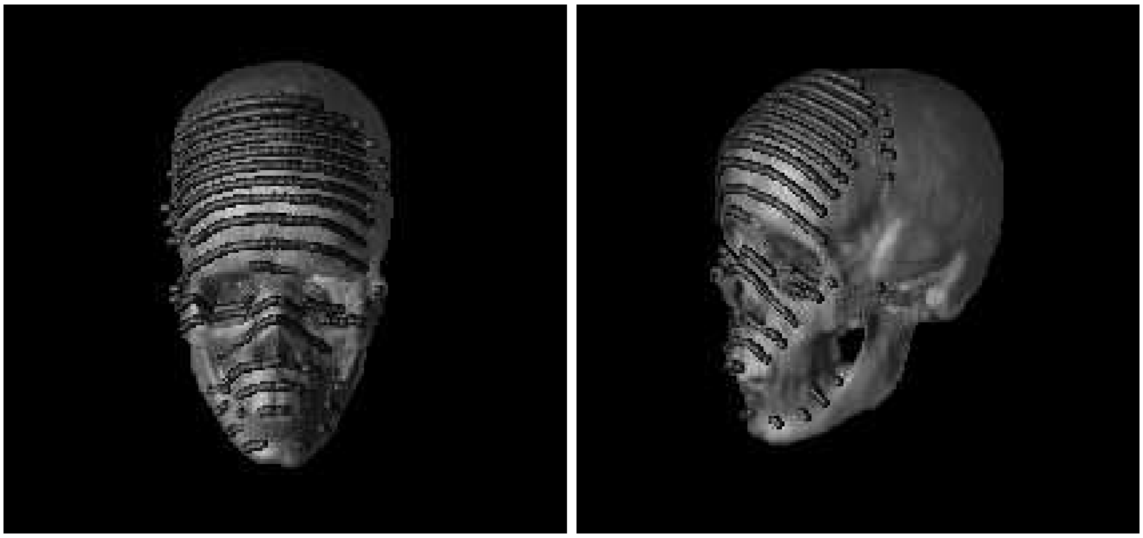


Figure 5-3: The coordinate registration problem in image guided surgery. The figures show two examples of the laser scan data (horizontal lines) overlaid on the MRI data (the skulls). The problem is to find the transformation that will align these two data sets.

5.2 Image guided surgery

There is recent study on frameless guidance systems to aid neurosurgeons in planning operations [Grimson et al., 1994]. A method has been developed for registering clinical data, such as segmented MRI or CT reconstructions, with the actual head of the patient, for which position and orientation information is provided by a laser scanning device. The data obtained from the scanning device is much coarser than that from MRI or CT. Figure 5-3 illustrates the two data sets overlaid. The detailed picture of the skull is obtained from its MRI scan, and the horizontal lines are the result of the laser scan. Furthermore, the coordinate frame of the laser device is different from the MRI device. To be able to register the two data sets, one needs to find the appropriate 3-dimensional transformation that will map them. At this point the problem can be seen as an optimization problem with six parameters. The parameters are the three distances and three angles of the transformation. The objective function measures the distance of the points in the laser scan data to their closest neighbors in the MRI data. The value returned is the mean squared distance.

In the project, Powell's method has been used for optimization. Figure 5-4 shows a comparison between Powell's method and Procedure 2-1. Powell's method fails to find a good solution if the initial configuration is too far from the global optimum. Procedure 2-1 not only finds better points, but also is able to do so from configurations that are further away.

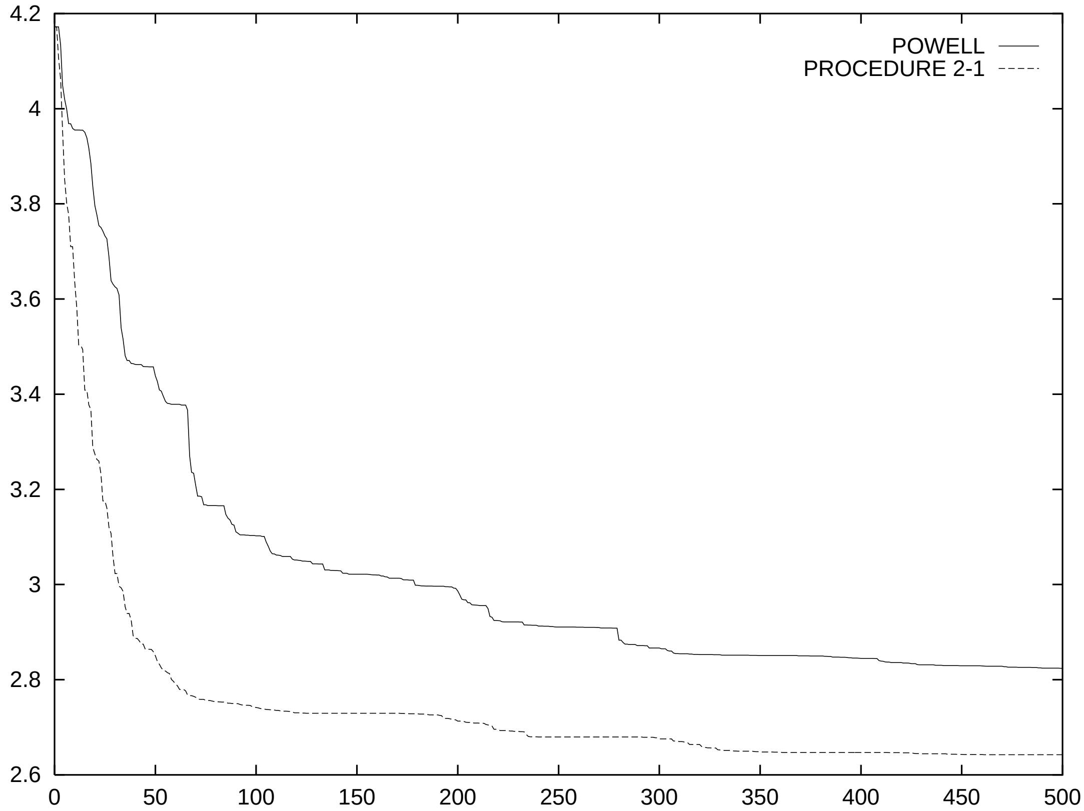


Figure 5-4: The comparison of Procedure 2-1 with Powell's method on the coordinate registration problem. The y axis shows the root mean squared distance in millimeters, the x axis is the number of function evaluations. The results are averaged over ten runs.

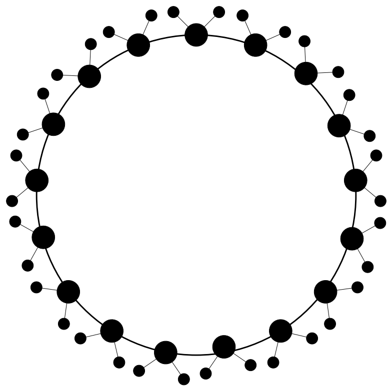


Figure 5-5: The Cycloheptadecane molecule has 51 atoms. The three coordinates for each atom are the input to the objective function. Thus the problem has 153 dimensions.

5.3 Molecule energy minimization

The conformation of a molecule is a description of its 3-dimensional structure. The physical properties and biological functions of organic molecules are intimately related to their conformations. Thus computing energetically stable conformations of organic molecules is an important problem. One way to compute the conformations is to minimize an energy function, but unfortunately, the energy function typically has many local minima.

Procedure 2-1 was tried on a small instance of this problem. Cycloheptadecane is a macrocyclic molecule. It consists of a chain of 17 carbons, where each carbon is connected to two hydrogen atoms as well as two other neighbor carbons. Figure 5-5 illustrates this structure. There are a total of 51 atoms in the molecule. Treating their x, y and z coordinates as separate parameters of the energy function, we have an optimization problem of 153 dimensions.

Recently, a novel method to solve this problem efficiently was proposed [Wang, 1994]. First, approximate solutions are found by combining smaller chains together. Then, these approximate solutions are further refined in an optimization loop. Figure 5-6 contains the comparison of Procedure 2-1 with Powell's method in this loop as well as a sample run starting from a random initial conformation.

Note that the gradient of the energy function typically can be computed. In that case an algorithm that can use this information such as conjugate gradient [Press et al., 1992, Polak, 1971], will be more efficient. The comparison is made with Powell's method because it is known as one of the most efficient algorithms among the ones that do not use the gradient.

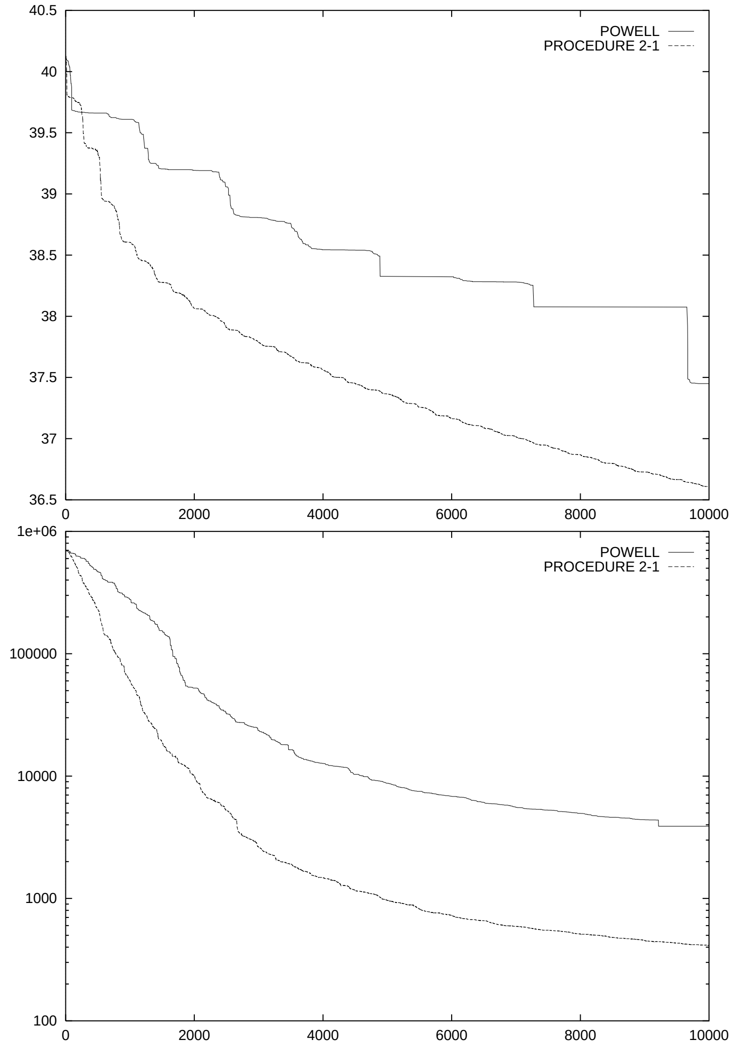


Figure 5-6: Comparison of Procedure 2-1 with Powell's method on the molecule energy minimization problem. The first example starts from a near optimal configuration. The second example starts with a random configuration. The lower lines are the traces of Procedure 2-1.

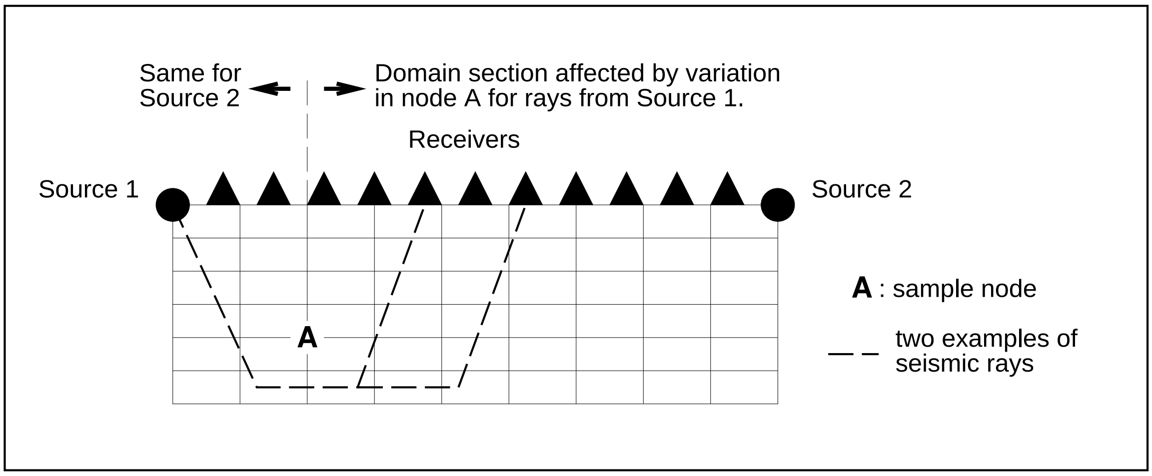


Figure 5-7: The problem of non-separability in refraction tomography. The variation in just the value at node A effects a part of the domain for one source and the remaining part for the other.

5.4 Refraction tomography

There is recent study in mathematical geophysics, on assessing the structure of sedimentary deposits using refraction seismic data [Williamson, 1990]. The synthetic example we have worked on is a rectangular section of earth that is 8000 meters long and 500 meters deep. There are 15 sources and 41 receivers for a total of 615 seismic rays. The path a ray follows depends on the seismic velocity structure in the 2-D section. The domain is divided into a rectangular grid and a seismic velocity value is defined at each node of this grid. In our example we used a grid of 9 by 5 nodes. The 45 seismic velocity values are the parameters of the objective function. The objective function is a ray-tracing algorithm. The algorithm simulates the propagation of the seismic rays from the sources to the receivers. The squared difference in time between the travel times of the simulated rays and a real data set is the output of the objective function.

This problem is a nice example of non-separability. A change in just one of the parameters will affect all of the domain on the other side of the source. This is illustrated in Figure 5-7. In this example, the sources are located at the extremities of the domain, thus a variation in just one parameter affects a part of the domain for one source and the remaining part for

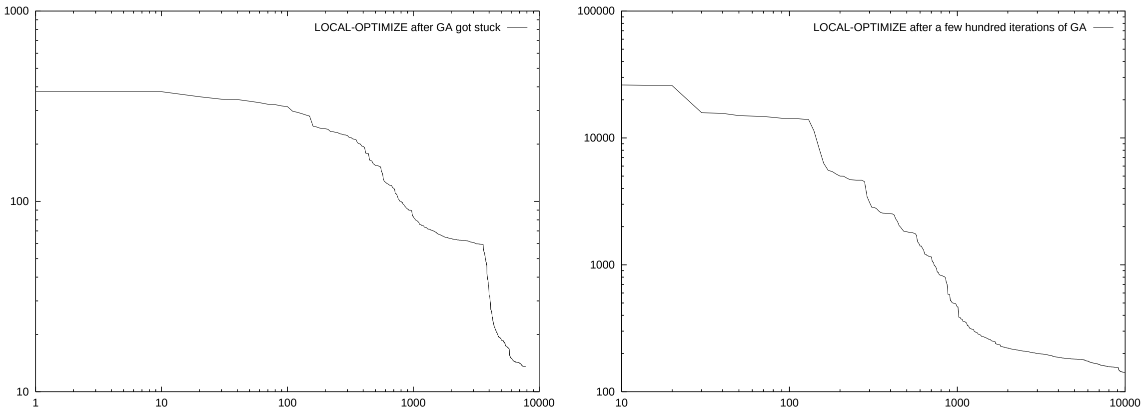


Figure 5-8: Two possible uses of LOCAL-OPTIMIZE. In the first example it is used to refine the search after the genetic algorithm was stuck. In the second example it is started after the genetic algorithm found a good region in a few hundred iterations

the other.

Several variations of genetic algorithms were tested on this problem. A genetic algorithm with a number of problem-specific modifications was found to give the best results. When my program was applied to the problem, it did not converge to points as good as the genetic algorithm in its first few iterations. This is probably due to the domain specific knowledge used in the other program. However, when the LOCAL-OPTIMIZE algorithm given in Procedure 4-1 was started at the points where the genetic algorithm got stuck, it was able to improve the solutions substantially. Furthermore, when LOCAL-OPTIMIZE was started after the genetic algorithm was run for a few hundred function evaluations, it was able to find better points more efficiently. Thus, a hybrid of the program with the domain-specific knowledge, and LOCAL-OPTIMIZE was found to give the best results.

The two examples shown in Figure 5-8 are different runs of the LOCAL-OPTIMIZE algorithm. The starting point for the first graph is the best solution found by the genetic algorithm with problem-specific modifications. The genetic algorithm discovered this solution in 13175 function evaluations and later got stuck. LOCAL-OPTIMIZE was started at that point and it was able to improve the error value about more than an order of magnitude. This example illustrates the difficulty with locating exact optima with the genetic algorithms. From the point the genetic algorithm got stuck, LOCAL-OPTIMIZE was able to find a way further down in the error space. This demonstrates the importance of being able to do a fine grained search by shrinking the step sizes.

For the second graph, the genetic algorithm was run for a few hundred iterations. At that point it was stopped, and Local-Optimize was started from the best point discovered so far. After about a thousand more function evaluations, the error was reduced to 370. This value is better than what the genetic algorithm got by itself in 13175 iterations.

The genetic algorithm is able to find good regions of the search space quickly by using problem-specific knowledge. However it is very inefficient in finding the optimum point in that region. The approach this suggests is to use problem-specific knowledge to locate the good regions in the space, and then to run Local-Optimize to find the exact location of the optima.

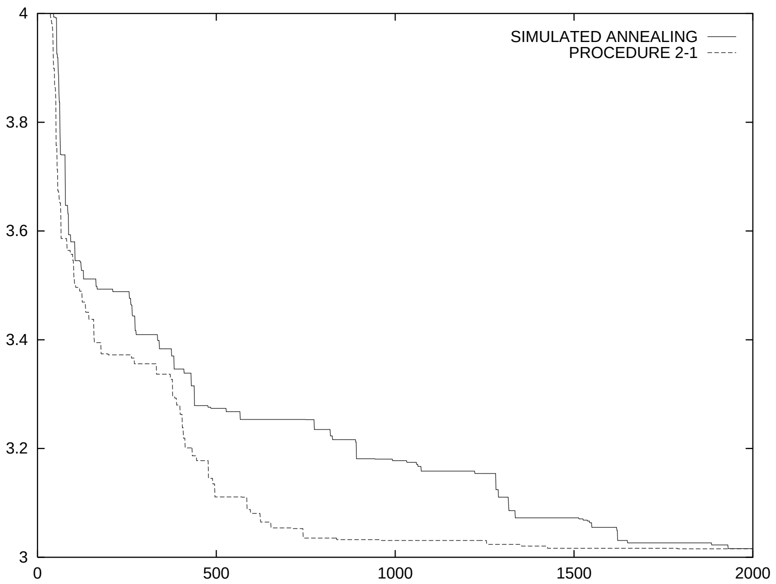


Figure 5-9: The performance of my algorithm compared to simulated annealing on the traveling salesman problem averaged over ten runs. The x axis shows the number of operations tried and the y axis shows the distance of the best discovered path.

5.5 The traveling salesman problem

The traveling salesman problem is the problem of finding the shortest cyclical itinerary for a traveling salesman who must visit each of N cities in turn. This problem has a very different nature from the above problems, it is an example of *combinatorial optimization*. There is an objective function to be minimized, but the search space is the discrete space of all permutations of N cities. The combinatorial optimization problems are challenging because the number of elements in the search space is factorially large, thus they cannot be exhaustively searched. Also it is hard to define a concept of "direction" in a permutation space.

The representation one chooses to use might make it possible to define concepts of "distance" and "direction" in non-cartesian spaces. One can carefully define various mutations such that "moving further in the same direction" is meaningful. It is also possible to define a notion of "size" in carefully selected operations. I will not present a detailed study of the application of my program on combinatorial problems. However, I thought it desirable to include one example, to show that the same key ideas apply even when one does not have a cartesian space.

I compared the results I obtained from a very simple implementation of my algorithm with results from simulated annealing. The simulated annealing program [Press et al., 1992] uses an efficient set of moves suggested by Lin [Lin, 1965]. The moves consist of two types: (a) A section of path is removed and then replaced with the same cities running in the opposite order; or (b) a section of path is removed and then replaced in between two cities on another, randomly chosen, part of the path. The objective function is the total length of the journey.

The implementation of Procedure 2-1 only uses the first type move, i.e. reversing a section of the path. The size of a step is defined as the width of the section reversed. The same principles of adjusting the step size are used.

Although this is a very different problem, the results of Procedure 2-1 are comparable to that of simulated annealing. Figure 5-9 shows the average over 10 runs for problems of 10 randomly selected city coordinates. These demonstrate that the main ideas presented in this thesis are not specific to numerical problems, but are general principles useful for optimization in general.

Chapter 6

Summary and future work

This thesis has described an optimization method that was designed to be robust and efficient. The ideas that built the method came from an inquiry into the nature and use of optimization programs based on genetic algorithms. The method uses information about the discovered local maxima to guide the search. The step sizes used are adaptive to the local terrain of the search space. Finally, the direction of most rapid ascent is computed from recent successful moves.

The method was tested on several problems and compared to other successful optimization algorithms. It has actively improved the solutions for real world problems selected from science, medicine and engineering. Its advantage is demonstrated on problems with high number of dimensions and many local maxima.

Theoretical work on the assumptions and performance of the method is required and the application to combinatorial problems need more study. Most importantly, more problems from different fields need to be found and studied in the context of the ideas presented in this thesis.

Appendix A

Sample code for Procedure 4.1

The procedure in the following page is the C implementation of the local optimization procedure given in Chapter 4. It has a probing vector that adapts its size and direction according to the local properties of the objective function. The constant NDIM is the number of dimensions, THRESHOLD is the minimum step size, INIT_SIZE is the initial step size. The variable *count* keeps the number of the calls to the objective function. The two arguments to the procedure are the objective function f and the initial starting point x. The procedure prints out successively better solutions and leaves the final solution in the argument vector x.

```
void local_optimize(double (*f)(), double x[])
{
  double u[NDIM], v[NDIM], xv[NDIM];
  double fx, fxv, vr;
  int vi, vvec, i, iter, maxiter;
  for(i=0; i<NDIM; i++) u[i] = v[i] = 0;
  vi = -1; vvec = 1; vr = -INIT_SIZE;
  fx = f(x); fxv = 1E10;
  printf("%d. %.4f <= ", count, fx);
  for(i=0; i<NDIM; i++) { printf("%.4f ", x[i]); }; printf("\n");
  while(fabs(vr) >= THRESHOLD) {
    maxiter = ((fabs(vr) < 2*THRESHOLD) ? 2*NDIM : 2);
    iter = 0;
    while((fxv >= fx) && (iter < maxiter)) {
      if(iter == 0) { for(i=0; i<NDIM; i++) xv[i] = x[i]; }
      else xv[vi] -= vr;
      if(vvec) vvec = 0;
      vr = -vr;
      if(vr > 0) vi = ((vi+1) % NDIM);
      xv[vi] += vr;
      fxv = f(xv);
      iter++;
    }
    if(fxv >= fx) {
      fxv = 1E10;
      vr /= 2;
    } else {
      fx = fxv; printf("%d. %.4f <= ", count, fx);
      for(i=0; i<NDIM; i++) { x[i] = xv[i]; printf("%.4f ", x[i]); }
      printf("\n");
      if(iter == 0) {
if(vvec) {
  for(i=0; i<NDIM; i++) { u[i] += v[i]; v[i] *= 2; xv[i] += v[i]; }
  vr *= 2;
} else {
  u[vi] += vr; vr *= 2; xv[vi] += vr;
}
fxv = f(xv);
      } else {
for(i=0; i<NDIM; i++) xv[i] += u[i];
xv[vi] += vr;
fxv = f(xv);
if(fxv >= fx) {
  for(i=0; i<NDIM; i++) { u[i] = 0; xv[i] = x[i]; }
  u[vi] = vr; vr *= 2;
  xv[vi] += vr; fxv = f(xv);
} else {
  for(i=0; i<NDIM; i++) x[i] = xv[i]; fx = fxv;
  u[vi] += vr;
  for(i=0; i<NDIM; i++) v[i] = 2*u[i]; vvec = 1;
  for(i=0; i<NDIM; i++) xv[i] += v[i]; fxv = f(xv);
  for(vr=0,i=0; i<NDIM; i++) vr += v[i]*v[i];
  vr = sqrt(vr);
}}}}}
```

Bibliography

[Bäck and Hoffmeister, 1992] Bäck, T. and Hoffmeister, F. (1992). *A User's Guide to GENEsYs 1.0*. University of Dortmund. Software package documentation.

[Cormen et al., 1990] Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*, chapter 17. The MIT Press, Cambridge, MA.

[De Jong, 1975] De Jong, K. (1975). *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan.

[de la Maza and Yuret, 1994] de la Maza, M. and Yuret, D. (1994). Dynamic hill climbing. *AI Expert*, 9(3).

[Dennis and Schnabel, 1983] Dennis, J. E. and Schnabel, R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ.

[Grimson et al., 1994] Grimson, W., Lozano-Pérez, T., III, W. W., Ettinger, G., White, S., and Kikinis., R. (1994). An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization. In *Computer Vision and Pattern Recognition Conference*, Seattle.

[Hillis, 1990] Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimizing procedure. *Physica*.

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

[Ingber, 1993] Ingber, L. (1993). *Adaptive Simulated Annealing (ASA)*. [ftp.caltech.edu: /pub/ingber/asa.Z]. Software package documentation.

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

[Lin, 1965] Lin, S. (1965). Computer solution of the tsp. *Bell System Technical Journal*, 44:2245–2269.

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

[Polak, 1971] Polak, E. (1971). *Computational Methods in Optimization*. Academic Press, New York.

[Powell, 1964] Powell, M. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7:155–162.

[Press et al., 1992] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in C: the art of scientific computing*, chapter 10. Cambridge University Press, New York, second edition.

[Teng, 1994] Teng, S. (1994). Voronoi diagrams. Lecture Notes for Computational Geometry, M.I.T.

[Wang, 1994] Wang, E. (1994). Conformational search of macrocyclic molecules. Masters thesis proposal, M.I.T.

[Williamson, 1990] Williamson, P. (1990). Tomographic inversion in reflection seismology. *Geophysical Journal International*.

[Winston, 1992] Winston, P. (1992). *Artificial Intelligence*, chapter 25. Addison-Wesley Publishing Company, Reading, MA, third edition.

[Yuret, 1992] Yuret, D. (1992). Evolution of evolution: an exploratory work on genetic algorithms. Undergraduate thesis, M.I.T.

[Yuret and de la Maza, 1993] Yuret, D. and de la Maza, M. (1993). Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 208–212.