RICE UNIVERSITY

Multi-Directional Search: A Direct Search Algorithm for Parallel Machines

by

Virginia Joanne Torczon

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

John E. Dennis, Jr., Chairman
Noah Harding Professor of Mathematical Sciences

Andrew E. Boyd
Assistant Professor of Mathematical Sciences

Kenneth W. Kennedy, Jr.
Noah Harding Professor in Mathematics in the Department of Computer Science

Richard A. Tapia
Professor of Mathematical Sciences

Houston, Texas

May, 1989

Abstract

Multi-Directional Search: A Direct Search Algorithm for Parallel Machines

by

Virginia Joanne Torczon

In recent years there has been a great deal of interest in the development of optimization algorithms which exploit the computational power of parallel computer architectures. We have developed a new direct search algorithm, which we call *multi-directional search*, that is ideally suited for parallel computation.

Our algorithm belongs to the class of direct search methods, a class of optimization algorithms which neither compute nor approximate any derivatives of the objective function. Our work, in fact, was inspired by the simplex method of Spendley, Hext, and Himsworth, and the simplex method of Nelder and Mead.

The multi-directional search algorithm is inherently parallel. The basic idea of the algorithm is to perform concurrent searches in multiple directions. These searches are free of any interdependencies, so the information required can be computed in parallel.

A central result of our work is the convergence analysis for our algorithm. By requiring only that the function be continuously differentiable over a bounded level set, we can prove that a subsequence of the points generated by the multi-directional search algorithm converges to a stationary point of the objective function. This is

of great interest since we know of few convergence results for practical direct search algorithms.

We also present numerical results indicating that the multi-directional search algorithm is robust, even in the presence of noise. Our results include comparisons with the Nelder-Mead simplex algorithm, the method of steepest descent, and a quasi-Newton method. One surprising conclusion of our numerical tests is that the Nelder-Mead simplex algorithm is not robust.

We close with some comments about future directions of research.

Acknowledgments

First, I would like to thank my family. Without their unwavering support I would never have made it this far. If they ever doubted that a historian could also be a mathematician, they never let it show.

I would like to thank the members of my committee; each one has made an important contribution to this work. Andy Boyd convinced first me, and then my chairman, that the last argument needed to complete the convergence proof was correct; he then pushed me to a most satisfying generalization. Ken Kennedy shared his knowledge of parallel computation and pointed me towards a very clever load balancing scheme. Richard Tapia deserves special credit for ever agreeing to accept me into the graduate program. He is also responsible for introducing me to optimization theory. Of late, he has shared his knowledge of the theory for descent methods, which has helped shaped the final form of my arguments. I have also enjoyed our many conversations on issues as diverse as music, minorities, and machismo.

I owe special thanks to my chairman, John Dennis. He asked me to stay, gave me this problem to work on, and made me tough. I owe a great deal to his inimitable style of motivation: His unerring response to my latest result would be either "You're wrong." or "That can't possibly be true." — which of course made me all the more determined to prove that I was, in fact, right. Now he moans that I am always right, as if there were no connection.

I would also like to thank the many other people at Rice from whom I have learned much: Richard Byrd helped isolate the last link needed to complete my convergence result. Linda Torczon and Keith Cooper have taught me much about computers and computation. Nancy Ginsburg shared the early, difficult years of graduate school.

Cathy Samuelsen took up where Nancy left off. Cathy and Karen Williamson suffered through numerous drafts of my thesis; it is better for their efforts.

Finally, I would like to thank my husband, Michael Lewis. He was the first, true believer in this work; I managed to convince him that I was right before I had even convinced myself. His faith in my mathematical abilities has pushed me further than anyone else thought possible. His good taste has left its mark on all this work.

Contents

Illustrations

Tables

Chapter 1

Introduction

Lately there has been great interest in developing optimization algorithms that take advantage of computational parallelism. We have made studies of a parallel multi-directional search algorithm to solve the general unconstrained optimization problem:

$$\text{Given } f : \mathbb{R}^n \to \mathbb{R}$$

$$\min_{x \in \mathbb{R}^n} f(x) \ .$$

Our goal has been to develop an algorithm with the following features:

- It conducts a multi-directional search that can be executed in parallel.
- It does not require information about the derivative of f.
- It is robust for problems of moderate size.
- It works well with "noisy" function values.
- It is easy to understand, easy to program, and easy to use.

We have now developed a new direct search method, which we call the multi-directional search algorithm, with these properties. Furthermore, we can prove the algorithm will converge to a minimizer of f under standard assumptions.

1.1 Background

The inspiration for our multi-directional search algorithm has come from the "direct search" methods for unconstrained optimization. The direct search methods are characterized by the fact that the decision-making process is based solely on function value information; these algorithms neither require nor estimate, in any direct sense, derivative information to determine a direction of descent. Several of the direct search methods — in particular the Nelder-Mead simplex algorithm [24] and the pattern search algorithm of Hooke and Jeeves [17] — have long enjoyed popularity in the community of computational scientists. These algorithms have remained popular for practical reasons. Often, particularly in experimental settings, derivatives are simply unavailable. In addition, function values based on experimental data are often "noisy" (i.e., they can only be trusted to a few digits of accuracy) so that finite-difference approximations may prove unreliable. Finally, most of the direct search methods are easy to understand, easy to program, and easy to use.

On the other hand, direct search methods have fallen out of favor with the numerical optimization community because, for the most part, they lack any convergence theory; they are exceedingly slow to converge, even in the neighborhood of a solution; and they either do not converge to a true solution, or else they converge *very* slowly, when the problem is "not small" — where by "not small" we mean ten or more variables. Despite these reservations, the active literature in such fields as analytical chemistry (e.g. [5], [7], [11], [13], [19], [23], [28], and [30]) suggests that practical considerations often win out over perceived disadvantages.

Our investigation of direct search methods deserves further comment, therefore, since it is a distinct departure from the current emphasis on the use of quasi-Newton methods in optimization. To preserve the sound theoretical properties of the sequential quasi-Newton methods, current parallel implementations of quasi-Newton methods consider only one search direction at each iteration. We though it would be of interest to focus, instead, on conducting a multi-directional search that could be executed in parallel. The direct search methods suggested a natural way to explore this approach.

Researchers in optimization generally agree that there are two main areas where improvement in the speed of execution for optimization algorithms can be made. First, since matrix operations contribute significantly to the cost of quasi-Newton procedures, immediate gains can be realized by employing advances made in parallel numerical linear algebra. In fact, Coleman and Li [12] found that the efficiency of their methods for solving systems of nonlinear equations on distributed memory multiprocessors depended very much on the efficiency of the linear algebra subroutines available for these machines. As a consequence, they devoted a great deal of their initial research effort to developing an efficient parallel triangular solver for a distributed memory multiprocessor [20], [21].

Second, since function evaluations constitute a significant portion of the cost in numerical optimization, one clear way to exploit parallel machines is to have different processors compute function values concurrently. Schnabel [31] suggested that when the evaluation of the function is expensive and the gradient is calculated by finite differences, then there are simple approaches to developing parallel unconstrained optimization algorithms. In this setting, a natural way to use parallel machines is to compute, in parallel, the n function evaluations required for the finite difference approximation to the gradient. This observation forms the basis for the subsequent work by Byrd, Schnabel and Shultz, [8] and [9], on parallel quasi-Newton methods for unconstrained optimization.

The key feature of the parallel algorithm we have developed is that it introduces a multi-directional search into the decision-making framework of the sequential Nelder-

Mead simplex algorithm. Rather than using $O(n)$ function evaluations to construct a finite difference approximation to the gradient for a single, albeit sophisticated, direction of search, why not use $O(n)$ function evaluations to search in n distinct directions? An examination of both the Nelder-Mead simplex algorithm and the pattern search algorithm of Hooke and Jeeves suggested a reasonable way to coordinate such a search. In addition, this approach did not require the solution of any linear systems of equations, which meant that there was no need to devote attention to parallel algorithms for numerical linear algebra. An additional feature of this approach, revealed by the convergence analysis, is that the multi-directional search algorithm contains a natural strategy for attacking the load balancing problem that has plagued much of the work being done to "parallelize" more conventional optimization methods.

Furthermore, we wished both to preserve the features of the direct search methods which make them so popular in certain circles, and to address the concerns raised by those who question the continued use of direct search methods. Thus we wanted an algorithm that did not require derivative information but was still robust. We wanted an algorithm that worked well in situations where the function values were noisy and yet was easy to understand and thus easy to implement. We also wanted an algorithm that could handle more than ten variables effectively. Finally, we wanted an algorithm that was backed by a convergence theory.

The result of our investigations is a multi-directional search algorithm which is a new direct search method that can be easily implemented on parallel machines — it is *not* a parallel implementation of an existing sequential direct search algorithm. In addition, we have developed a convergence theory for this algorithm that we believe can be extended to several of the original direct search algorithms for which we know of no prior convergence results. Finally, we have evidence to suggest that the multi-directional search algorithm also preserves many of the performance features that we were interested in maintaining.

1.2 Choice of search direction and step length

The multi-directional search algorithm could be considered a descent method that uses a line search as a globalization strategy. This characterization of the algorithm will become apparent in Chapter 3 when we discuss the convergence analysis for the multi-directional search algorithm. Given this characterization of the algorithm, there are two issues which must be addressed:

- The algorithm must include a method for choosing a search direction that will guarantee a decrease in the function value, at the current iterate, for a step of the appropriate length.
- The algorithm must include a method for finding the appropriate step length.

The archetype of descent methods, the method of steepest descent, uses the gradient to resolve both of these issues. If the given point is not a local minimizer, the negative gradient provides a search direction for which descent is guaranteed if the step taken in that direction is sufficiently small. In addition, the gradient describes the rate of change of the function from the given point, which allows for a quantitative notion of "sufficient decrease" to prevent steps that are either too long or too short. In the first case, the average rate of decrease in the function value from the current iterate to the next iterate must be at least some prescribed fraction of the initial rate of decrease in that direction. In the second case, the rate of decrease of the function f in the search direction at the new iterate must be larger than some prescribed fraction of the rate of decrease in the search direction at the the current iterate. These two requirements for sufficient decrease are known as the Armijo-Goldstein conditions.

More sophisticated line search algorithms incorporate curvature information (second derivatives) to help determine a search direction which should produce a decrease in the function value at the next iterate. The procedure for choosing the length of the step may or may not make further use of the Hessian, but the gradient is still needed to enforce the notion of sufficient decrease.

Having relinquished derivative information for practical considerations, the question now becomes: How do we find both a descent direction and a step of the appropriate length? To determine the search direction, some direct search methods consider a natural alternative: at every iteration they explore each direction in a set of n linearly independent search directions, where n is the dimension of the domain. If the function is differentiable, then at least one of these directions is not orthogonal to the gradient and therefore determines a descent direction. Again, if a step taken in this direction is small enough, descent is guaranteed. This observation suggests a natural way to develop algorithms that search multiple directions concurrently. Less clear is how to determine a step of the appropriate length. The direct search simplex methods, first introduced by Spendley, Hext and Himsworth [32] and then modified by Nelder and Mead [24], provide a simple mechanism to determine the size of the step to be taken. A judicious use of these two ideas introduces enough structure to derive a convergence proof for the resulting algorithm.

Chapter 2

Algorithm

2.1 A description of the multi-directional search algorithm

At any iteration k, where $k \geq 0$, the multi-directional search algorithm requires $n+1$ points, $v_0^k, \ldots, v_n^k$, which define a nondegenerate simplex in $I\!R^n$. By nondegenerate we mean that the set of n edges adjacent to any given vertex in the simplex spans $I\!R^n$. In Figure 2.1 this is the triangle formed by the vertices $< v_0^k, v_1^k, v_2^k >$. The edges of the simplex are used to define the search directions, the orientation of the search directions, and the step size in each direction.

We begin by computing the function values at all $n+1$ vertices in the original simplex. Using this function information, the algorithm distinguishes the "best" vertex in the simplex, where the "best" vertex is defined to be the vertex having the smallest function value. We follow the convention that v_0^k denotes the best vertex at the current iteration k. We then have a best vertex, v_0^k, which satisfies:

$$f(v_0^k) \quad \leq \quad f(v_i^k) \quad \text{for} \quad i = 1, \ldots, n.$$

The n edges connecting the best vertex to the remaining n vertices determine a set of n linearly independent search directions. In Figure 2.1 these would be the edges $\overline{v_0^k v_1^k}$ and $\overline{v_0^k v_2^k}$.

2.1.1 The reflection step

The algorithm now takes a step from the best vertex, v_0^k, in each of these n directions. The length of each step is equal to the length of the edge that determined the search direction. Geometrically, this could be viewed as "reflecting" the original simplex

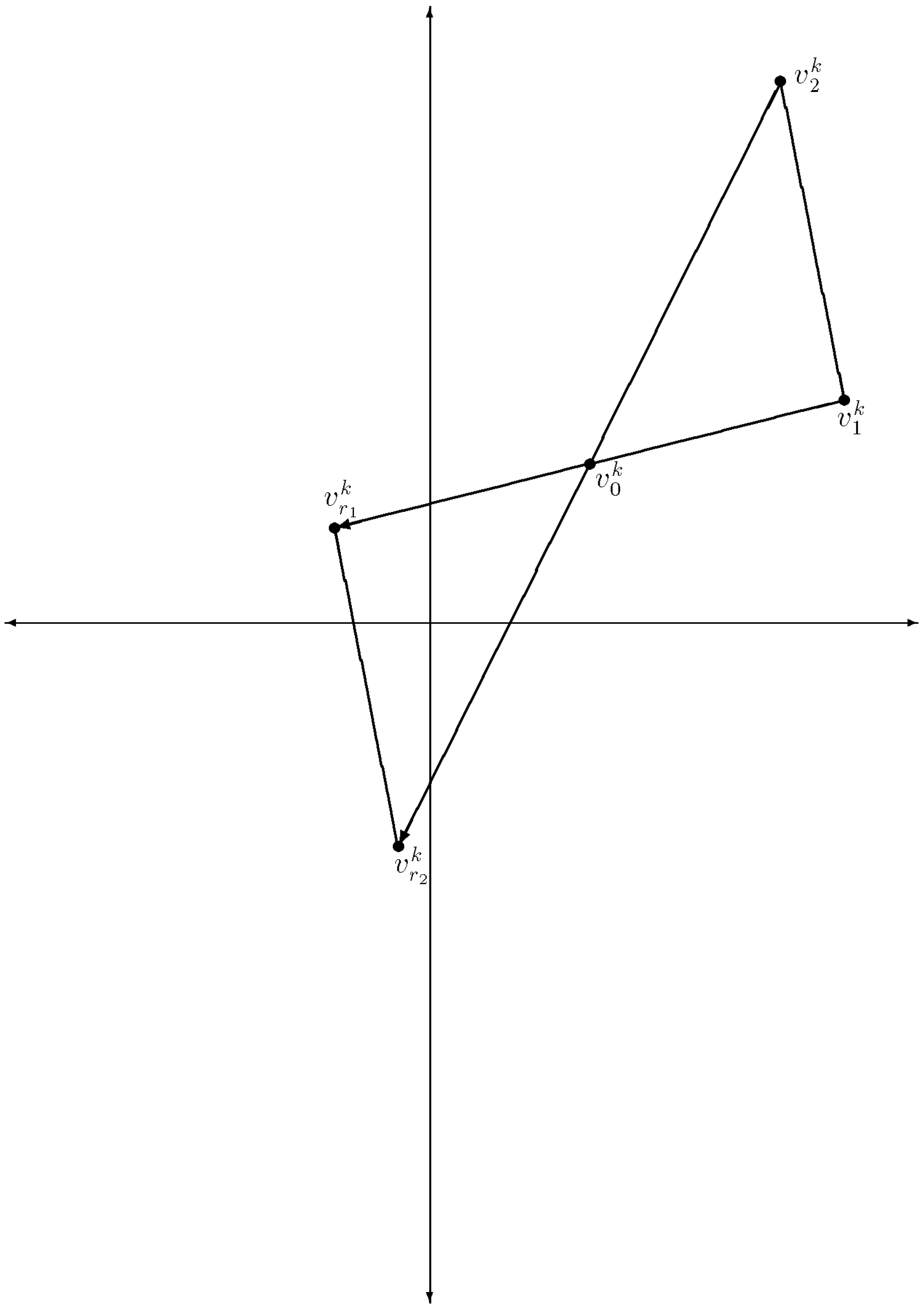

Figure 2.1 The original simplex and its reflection

through the best vertex to give a new simplex. The result, as can be seen in Figure 2.1, is a trial simplex, $< v_0^k, v_{r_1}^k, v_{r_2}^k >$, that shares only the best vertex, v_0^k, with the original simplex, $< v_0^k, v_1^k, v_2^k >$. The angles in the trial simplex, and the lengths of all the edges, are the same as those in the original simplex.

The step is deemed successful if it satisfies the following simple acceptance test: Does one of the new vertices have a function value that is better than the function value at the best vertex in the original simplex? Specifically, can the following condition be satisfied:

$$\min\left\{ f(v_{r_i}^k),\ i = 1, \ldots, n \right\} \ < \ f(v_0^k) \ ? \tag{2.1}$$

The reason for this acceptance test is straightforward. We would like the reflected simplex to produce a new best vertex; i.e., a vertex with a function value that is less than the function value at the current best vertex, v_0^k. As can be seen in Figure 2.1, if the best vertex is not replaced, then at the next iteration the new vertices, $v_{r_1}^k$ and $v_{r_2}^k$, will be reflected through the same best vertex, v_0^k, to restore the original simplex, $< v_0^k, v_1^k, v_2^k >$. Checking the acceptance criterion requires the calculation of the function values at the n new vertices, $v_{r_1}^k, \ldots, v_{r_n}^k$. (Note that there are no data dependencies, so these function values are easily computed in parallel.) Now there are two possibilities: The acceptance test for the reflection step either is or is not satisfied. In either case we will consider a new trial simplex, as described in the next two sections.

2.1.2 The expansion step

Suppose that one of the new vertices does satisfy the acceptance criterion stated in (2.1). The logical question is then whether or not further improvements could be found by considering an even longer step size. To answer this question, the algorithm

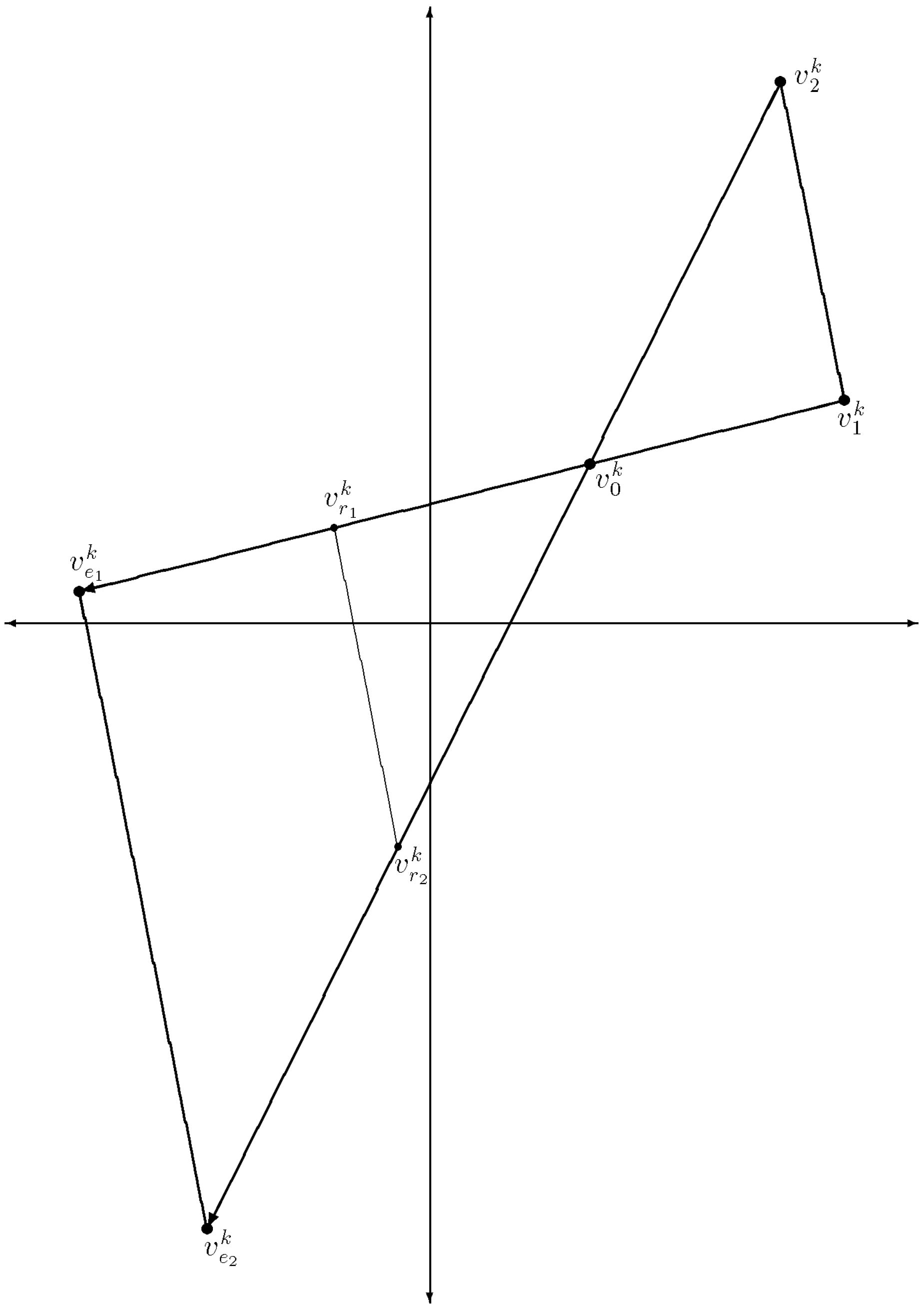

Figure 2.2 The original simplex with the reflected simplex and its expansion

again takes a step from the best vertex, v_0^k, in each of the original n search directions, but now each step is twice as long. Geometrically, as shown in Figure 2.2, the algorithm "expands" the trial simplex by doubling the length of every edge in the reflected simplex to give the new trial simplex $< v_0^k, v_{e_1}^k, v_{e_2}^k >$. Note that while the angles in the new trial simplex are still the same as the angles in the original simplex, the lengths of all the edges have been rescaled by a factor of two.

Acceptance of the expanded simplex is based on the reasoning that it is not necessary to double the step length unless such a step improves on the decrease seen in the function values found with the original step length. This leads to the following acceptance condition for the expansion step:

$$\min\left\{f(v_{e_i}^k),\ i=1,\ldots,n\right\} \ < \ \min\left\{f(v_{r_i}^k),\ i=1,\ldots,n\right\}. \tag{2.2}$$

When the expanded simplex is accepted, the next iteration begins with the simplex $< v_0^k, v_{e_1}^k, \ldots, v_{e_n}^k >$. Otherwise, we note that we only considered the expansion step after we had verified that the reflected simplex satisfied the acceptance condition (2.1). Thus, when we consider but do not accept the expanded simplex we simply begin the next iteration with the reflected simplex $< v_0^k, v_{r_1}^k, \ldots, v_{r_n}^k >$.

2.1.3 The contraction step

If none of the new vertices in the reflected simplex satisfies the acceptance criterion given in (2.1), the logical question now becomes: Were the steps we considered too long? A natural response to this question is to restart the search with shorter step sizes. Towards this end, the algorithm halves the lengths of the steps that can be taken. Geometrically, as can be seen in Figure 2.3, the algorithm simply "contracts" the original simplex towards the best vertex, v_0^k, by halving every edge in the sim-

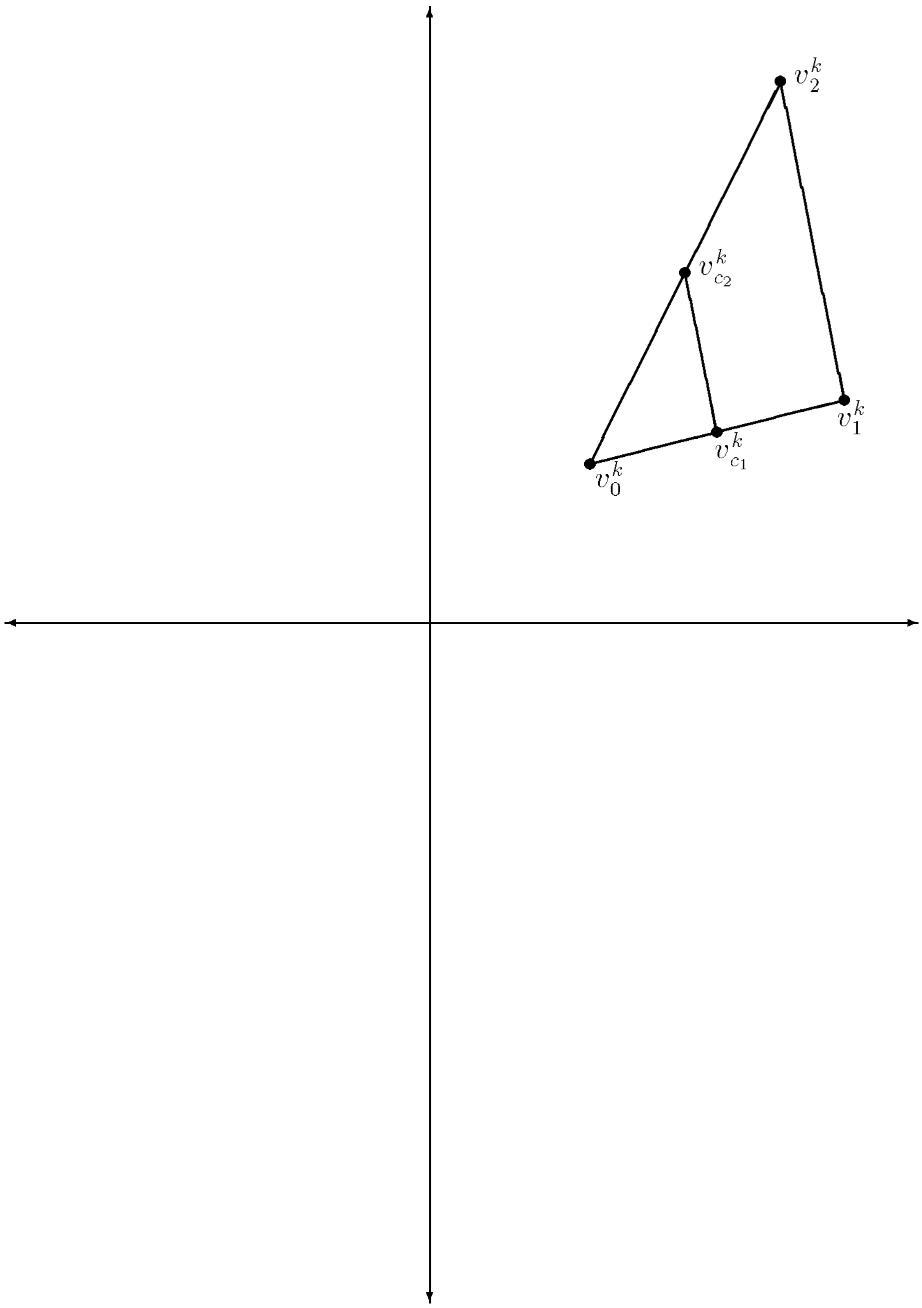

Figure 2.3 The original simplex and its contraction

plex. The algorithm will then start the next iteration with the contracted simplex $< v_0^k, v_{c_1}^k, \ldots, v_{c_n}^k >$.

Before proceeding to the next iteration, though, we check to see whether or not one of the n new vertices in the contracted simplex will satisfy the simple decrease condition:

$$\min\left\{f(v_{c_i}^k),\ i=1,\ldots,n\right\} \;<\; f(v_0^k)\,. \tag{2.3}$$

We would still like to find a new best vertex. We compute the function values at the vertices in the contracted simplex, and then check our simple decrease condition. If condition (2.3) is satisfied, the algorithm starts the next iteration with a new best vertex. If not, at the next iteration the algorithm will again be searching from the same best vertex, v_0^k, in each of the same n directions considered in the previous iteration. Now, however, the length of each step has been halved. In either case, if the contracted simplex is constructed, it is the simplex used to start the next iteration.

2.2 New search directions and new step lengths

The goal of the multi-directional search algorithm is to construct a sequence of best vertices, $\{v_0^k\}$, that converges to a critical point — ideally a minimizer — of the function. For this reason we require that the sequence of function values at the best vertex, $\left\{f(v_0^k)\right\}$, be monotonically decreasing. Thus, we only accept a new best vertex if it satisfies the simple decrease condition, as specified in (2.1), (2.2), and (2.3).

There are practical reasons for wishing to replace the best vertex. The choice of a best vertex determines the choice of search directions since the search directions are defined by the n edges adjacent to the best vertex. As we have already noted, it is imperative that we not accept the reflected simplex unless we can replace the best vertex or else we are doomed to flip back and forth between the original simplex

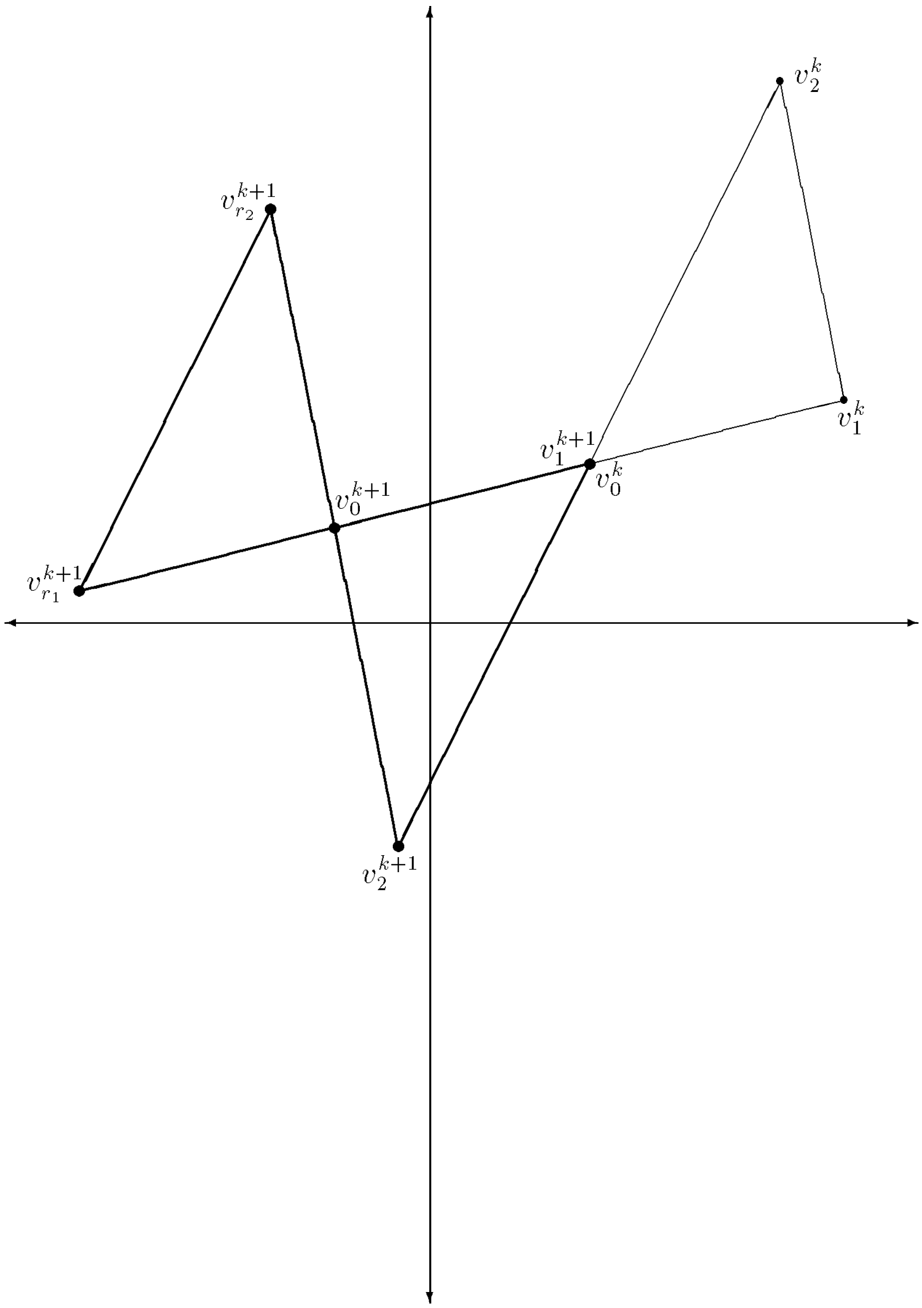

Figure 2.4 The next iteration — with new search directions

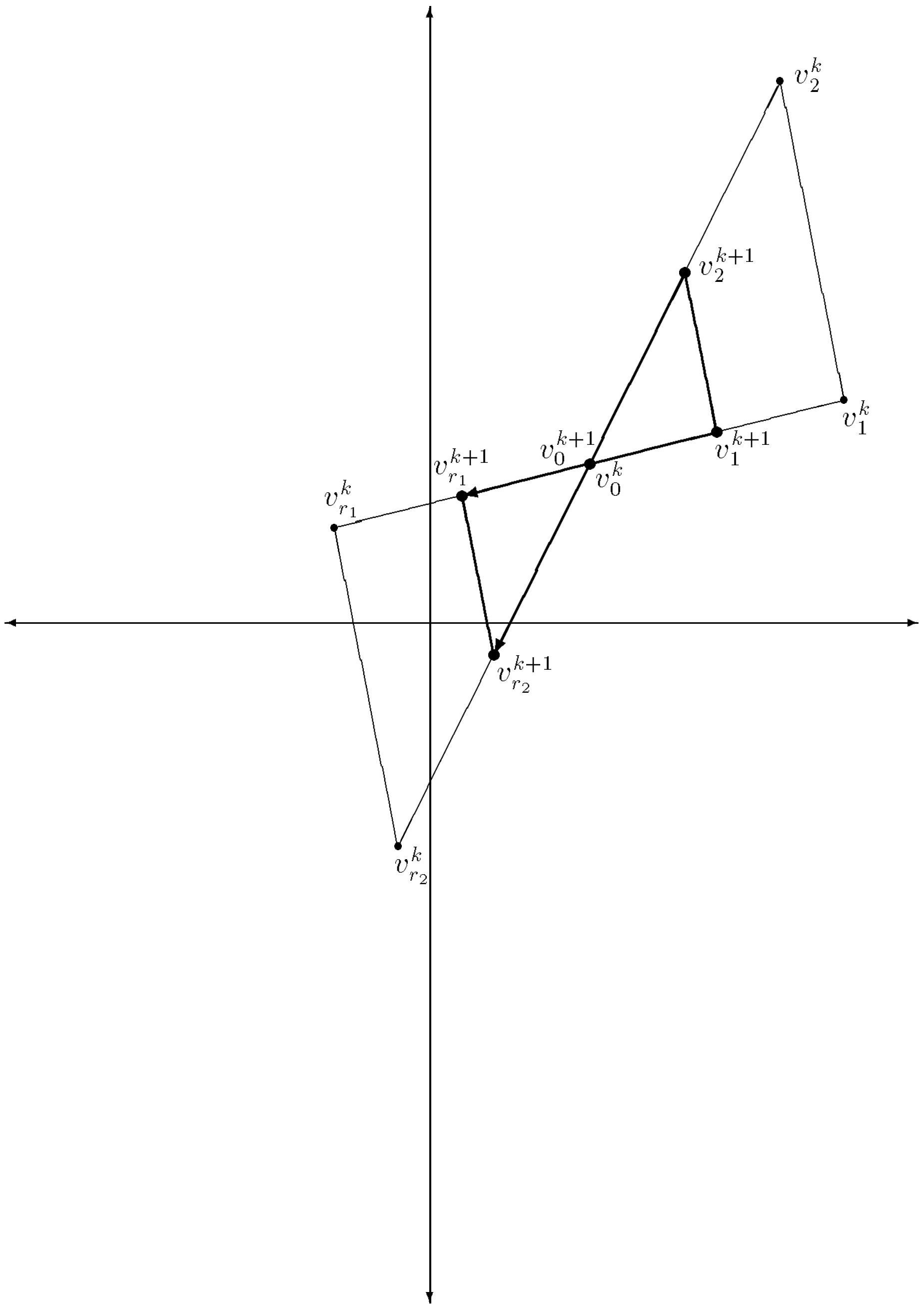

Figure 2.5 The next iteration — with new step sizes

and its reflection. If we do accept either the reflected or expanded simplex, however, we are guaranteed to begin the next iteration with a new best vertex and $n-1$ new search directions, as can be seen in Figure 2.4. Note that we retain one direction from the previous iteration — the direction which gave us our new best vertex. This means that we do not discard a direction along which we have seen descent until we have seen further descent in another direction. This is particularly satisfying when we accept the expanded simplex because we have seen significant decrease in one of the directions. Then, even though we do not continue to search along that direction in the current iteration, we do include that direction in our search at the next iteration.

If we do not replace the best vertex, and thus do not replace the set of search directions, we try a different strategy: We change the size of the step we take. First, we halve the size of the current simplex; this has the effect of halving the size of the reflection simplex in the next iteration. The result can be seen in Figure 2.5. This process of halving the step sizes continues until the simple decrease condition, either (2.1) or (2.3), has been satisfied.

Having made these observations, we are now ready to give a formal statement of the algorithm.

2.3 The multi-directional search algorithm

Given an initial simplex, S_0, with vertices $< v_0^0, v_1^0, \ldots, v_n^0 >$, $\mu \in (1, +\infty)$, and $\theta \in (0, 1)$.

$min \leftarrow \arg_i \min\{f(v_i^0),\ i = 0, \ldots, n\}$

swap v_{min}^0 and v_0^0

for $k = 0, 1, \ldots$

Check the stopping criterion.

for $i = 1, \ldots, n$ /* reflection step

$v_i^{k+1} \leftarrow 2\, v_0^k - v_i^k$

calculate $f(v_i^{k+1})$

if $\left(\min\left\{f(v_i^{k+1}),\ i = 1, \ldots, n\right\} < f(v_0^k)\right)$ **then**

for $i = 1, \ldots, n$ /* expansion step

$v_{e_i}^k \leftarrow (1 - \mu)\, v_0^k + \mu\, v_i^{k+1}$

calculate $f(v_{e_i}^k)$ for $i = 1, \ldots, n$

if $\left(\min\left\{f(v_{e_i}^k),\ i = 1, \ldots, n\right\} < \min\left\{f(v_i^{k+1}),\ i = 1, \ldots, n\right\}\right)$

$v_i^{k+1} \leftarrow v_{e_i}^k$ for $i = 1, \ldots, n$

else

for $i = 1, \ldots, n$ /* contraction step

$v_i^{k+1} \leftarrow (1 + \theta)\, v_0^k - \theta\, v_i^{k+1}$

calculate $f(v_i^{k+1})$ for $i = 1, \ldots, n$

endif

$min \leftarrow \arg_i \min\left\{f(v_i^{k+1}),\ i = 1, \ldots, n\right\}$

if $\left(f(v_{min}^{k+1}) < f(v_0^k)\right)$ swap v_{min}^{k+1} and v_0^k

Note that in the description of the algorithm we assumed that the contraction factor, θ, was equal to one half, while the expansion factor, μ, was equal to two. The choice of θ and μ, as well as such issues as producing an initial simplex and deciding when to stop the algorithm, will be discussed in Chapter 4.

2.4 Discussion

By allowing the simplex to expand and contract, the algorithm allows for both longer and shorter steps. Note, however, that the actual step size depends on the choice of the initial simplex since the lengths of the edges in the initial simplex determine relative step lengths of size one, while the scaling factors for the expansion and contraction are fixed across all iterations. The edges in the initial simplex also determine a finite set of fixed search directions that are maintained across all iterations. Thus, the progress of the procedure depends, to a large extent, on the choice of the initial simplex.

There are still two questions that we need to answer. Are these step sizes flexible enough to allow us to satisfy the strict decrease requirement enforced by the acceptance criterion given in (2.1)? The answer is "yes." If the function is differentiable, and the best vertex is not at a critical point of the function, repeatedly contracting the simplex will lead, in a finite number of iterations, to a simplex that satisfies the acceptance criterion. Will this process converge to a solution? The answer is again "yes," as will be discussed in the next chapter.

Chapter 3

Convergence

3.1 The convergence theorem

Before proceeding to a formal statement of the theorem, let us first set the stage. We begin by restricting our attention to the sequence $\left\{v_0^k\right\}$, which is the sequence of best vertices generated by the multi-directional search algorithm. Recall that at every iteration k, v_0^k is chosen to be a vertex for which

$$f(v_0^k) \quad \leq \quad f(v_i^k) \quad \text{for} \quad i = 1, \ldots, n.$$

We will define the level set of f at v_0^0, where v_0^0 is the best vertex of the initial simplex, to be

$$L(v_0^0) \quad = \quad \left\{x : f(x) \leq f(v_0^0)\right\}.$$

Given $y \in I\!R^n$, we define the contour $C(y)$ to be

$$C(y) \quad = \quad \{x : f(x) = f(y)\}.$$

We require that the function f be continuously differentiable, even though the algorithm does not explicitly compute derivatives. Having made this assumption we then define X_* to be the set of stationary points of the function f in $L(v_0^0)$:

$$X_* \quad = \quad \left\{x \in L(v_0^0) : \nabla f(x) = 0\right\}.$$

Finally, we require that the expansion and contraction factors, μ and θ, be rational.

We are now ready for a formal statement of the theorem and its corollary.

Theorem 3.1 Suppose that (1) f is continuously differentiable, and (2) $L(v_0^0)$ is compact. Then some subsequence of $\left\{v_0^k\right\}$ converges to a point $x_* \in X_*$. Furthermore, $\left\{v_0^k\right\}$ converges to C_*, where $C_* = C(x_*)$, in the sense that

$$\lim_{k\to\infty} \left[\inf_{x\in C_*} \left\|v_0^k - x\right\|\right] \;=\; 0\,.$$

Corollary 3.1 Suppose that (1) f is continuously differentiable, (2) $L(v_0^0)$ is compact, and (3) f is strictly convex on $L(v_0^0)$. Then

$$\lim_{k\to\infty} v_0^k \;=\; x_*\,,$$

where x_* is the unique minimizer of f on $L(v_0^0)$.

A formal proof of Theorem 3.1 is given in Section 3.3. The next section will build the machinery necessary to establish the convergence result.

3.2 Descent methods

The most natural strategy for any minimization algorithm is to ensure that each step accepted by the algorithm decreases the value of the function f; i.e., at every iteration k, we require that the algorithm produce an iterate x^{k+1} such that

$$f(x^{k+1}) \;<\; f(x^k)\,. \tag{3.1}$$

Methods which satisfy this condition are called *descent methods*. Much of the convergence analysis to be found in the standard optimization literature is built around this simple requirement. The method of steepest descent, as first defined by Cauchy, is the archetypal descent method — as its very name would suggest. Newton's method, without modification, is not a descent method; however, considerable effort has been devoted to developing modifications of Newton's method so as to guarantee that condition (3.1) is satisfied. Not too surprisingly, then, there exists a rich understanding of descent methods and the conditions under which they are guaranteed to converge to a stationary point of a function. For a further discussion of descent methods, see, for example, either Dennis and Schnabel [14], Gill, Murray, and Wright [16], or Ortega and Rheinboldt [25].

Our analysis of the convergence properties of the multi-directional search algorithm begins with the observation that the algorithm is, in fact, a descent method. This is not immediately obvious given the above definition of a descent method. However, if we restrict our attention to the sequence of best vertices we can show that if v_0^k is not a stationary point of f, then there exists a strictly positive integer p_k, which depends on k, such that

$$f(v_0^{k+p_k}) \quad < \quad f(v_0^k)\,. \tag{3.2}$$

We revise the original definition of a descent method to accommodate the fact that the multi-directional search algorithm does not produce a *new* best vertex at every iteration. Instead, we note that since the best vertex is not replaced until we find a vertex which produces decrease on the function value at the best vertex, we are guaranteed that after every iteration, either

$$f(v_0^{k+1}) \;<\; f(v_0^k) \qquad \text{or} \qquad v_0^{k+1} \;=\; v_0^k\,.$$

In the first case, we have a *new* best vertex. In the second case, we simply start the next iteration with the same best vertex, but smaller step sizes. The question now becomes: Can the multi-directional search algorithm produce a *new* best vertex in a finite number of iterations? To verify this condition we consider each of the three possible steps the algorithm can take: the reflection step, the expansion step, and the contraction step.

3.2.1 Guaranteeing strict decrease

To accept the reflection step, the multi-directional search algorithm requires that the reflected simplex produce at least one vertex which satisfies the simple decrease condition

$$f(v_{r_i}^k) \;<\; f(v_0^k)\,. \tag{3.3}$$

If the reflected simplex is accepted, then at the next iteration $v_{r_i}^k$ will become the new best vertex, i.e.,

$$v_0^{k+1} \;\leftarrow\; v_{r_i}^k$$

and so

$$f(v_0^{k+1}) \;<\; f(v_0^k)\,. \tag{3.4}$$

Thus we satisfy our strict decrease condition (3.2) with $p_k = 1$.

We do not even consider the expansion step unless we have already found a vertex in the reflected simplex for which the simple decrease condition (3.3) is satisfied. The acceptance test for the expansion step is even more rigorous: we only accept the expanded simplex if one of its vertices produces even further decrease in the function value. So again, we have guaranteed that the strict decrease condition (3.2) has been satisfied with $p_k = 1$.

The contraction step is the most interesting case. Recall that the algorithm only considers the contracted simplex after the reflected simplex has been rejected because it did not produce a vertex that satisfied our simple decrease criterion (3.3). We compute the function values at each of the new vertices in the contracted simplex and check to see if we have indeed found a vertex, $v_{c_i}^k$, which satisfies the simple decrease condition

$$f(v_{c_i}^k) \;<\; f(v_0^k)\,. \tag{3.5}$$

If so, we start the next iteration with a new best vertex, since we make the replacement

$$v_0^{k+1} \leftarrow v_{c_i}^k .$$

But if we do not satisfy the simple decrease condition (3.5), then we start the next iteration with the current best vertex; i.e.,

$$v_0^{k+1} \leftarrow v_0^k .$$

The question is this: If we repeatedly contract the simplex, will we eventually produce a new best vertex — one which satisfies condition (3.2)? The answer is "yes" — if we assume that the function is differentiable and that v_0^k is not a stationary point.

Note that by definition the n edges adjacent to v_0^k form a set that is linearly independent. We assume that v_0^k is not a stationary point. Then, since the set of edges spans $I\!R^n$, we are guaranteed at least one v_i^k, for $i = 1, \dots, n$, such that the edge $\overline{v_0^k v_i^k}$ is not orthogonal to the gradient of f at v_0^k. Then either this edge, or its reflection, determines a direction of descent. Recall that if the current best vertex is not replaced by one of the vertices in the contracted simplex, then $v_0^{k+1} \leftarrow v_0^k$ and $v_i^{k+1} \leftarrow v_{c_i}^k$, for $i = 1, \dots, n$, as we first observed in Figure 2.5. This means that at the next iteration we will be searching along the same set of search directions. But then $v_{r_i}^{k+1}$ will be contained in the edge $\overline{v_0^k v_{r_i}^k}$, as can be seen in Figure 3.1. Thus we are guaranteed to have at least one direction which produces descent since the algorithm searches back and forth along both the edge $\overline{v_0^k v_i^k}$, in the original simplex, and its corresponding edge, $\overline{v_0^k v_{r_i}^k}$, in the reflected simplex.

Since we are guaranteed that at least one of the n edges adjacent to the best vertex identifies a direction of descent, we need only show that the algorithm finds a step, in a finite number of iterations, which produces decrease in the function value at the best vertex. To see that this happens, we simply note that we are generating

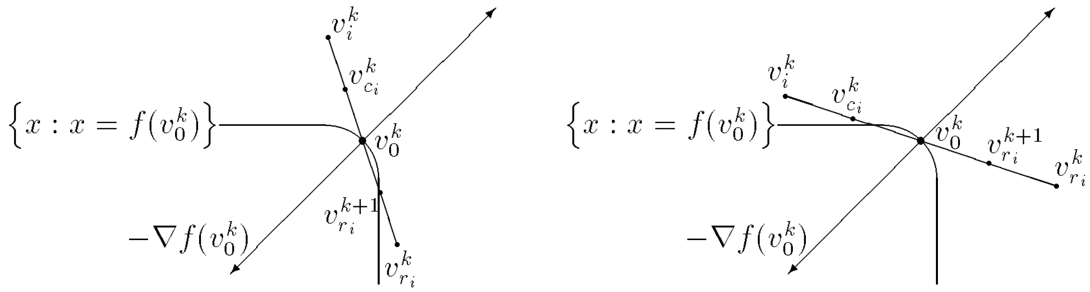

Figure 3.1 The two cases which guarantee a direction of descent

two sequences, the contracted vertices $\{v_{c_i}^k\}$ and the reflected vertices $\{v_{r_i}^k\}$, both of which are converging to v_0^k. If v_0^k is not a stationary point, the differentiability of f then ensures that one of these two sequences will produce a new best vertex in a finite number of iterations. Hence, we can guarantee that condition (3.2) is satisfied: there exists a subsequence of $\left\{f(v_0^k)\right\}$ which is strictly decreasing. This argument is formalized in Lemma 3.1.

> **Lemma 3.1** Suppose that (1) f is differentiable and (2) v_0^k is not a stationary point. Then there exists a strictly positive integer p_k, which depends on k, such that
>
> $$f(v_0^{k+p_k}) < f(v_0^k).$$
>
> This means that the sequence $\left\{v_0^k\right\}$ has a subsequence $\left\{v_0^{k_j}\right\}$ such that for all j
>
> $$f(v_0^{k_{j+1}}) \;<\; f(v_0^{k_j}).$$

Proof By definition, the set of edges adjacent to any vertex in a simplex is linearly independent. Thus, the set of n edges adjacent to current best vertex, by which we really mean the set of vectors

$$\left\{\left(v_i^k - v_0^k\right) : i = 1, \ldots, n\right\},$$

spans $I\!R^n$.

By assumption, f is differentiable and v_0^k is not a stationary point. Thus, we can conclude that there exists at least one i, for $i = 1, \ldots, n$, such that

$$\nabla f(v_0^k)^T (v_i^k - v_0^k) \quad \neq \quad 0\,.$$

There are then two cases to consider:

- Case 1: $\nabla f(v_0^k)^T (v_i^k - v_0^k) > 0$
- Case 2: $\nabla f(v_0^k)^T (v_i^k - v_0^k) < 0$.

Case 1: $\nabla f(v_0^k)^T (v_i^k - v_0^k) > 0$

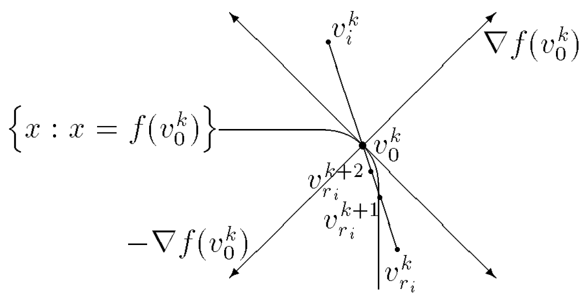

Figure 3.2 Case 1: $\nabla f(v_0^k)^T (v_i^k - v_0^k) > 0$

Since f is differentiable, we consider the directional derivative of f at v_0^k in the direction $(v_{r_i}^k - v_0^k)$:

$$\nabla f(v_0^k)^T (v_{r_i}^k - v_0^k) \quad = \quad \lim_{h \to 0} \frac{f\left(v_0^k + h\,(v_{r_i}^k - v_0^k)\right) - f(v_0^k)}{h}\,.$$

Since $\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k) < 0$ then

$$\lim_{h\to 0} \frac{f\left(v_0^k + h\,(v_{r_i}^k - v_0^k)\right) - f(v_0^k)}{h} \quad < \quad 0\,.$$

Thus, there exists an $h > 0$ such that

$$f\left(v_0^k + h\,(v_{r_i}^k - v_0^k)\right) \quad < \quad f(v_0^k)\,.$$

Case 2: $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$

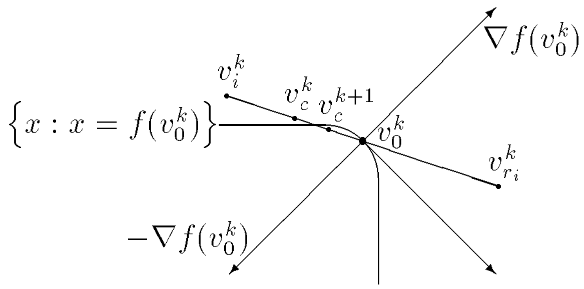

Figure 3.3 Case 2: $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$

Since f is differentiable, we consider the directional derivative of f at v_0^k in the direction $(v_i^k - v_0^k)$:

$$\nabla f(v_0^k)^T(v_i^k - v_0^k) \quad = \quad \lim_{h\to 0} \frac{f\left(v_0^k + h\,(v_i^k - v_0^k)\right) - f(v_0^k)}{h}\,.$$

Since $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$ then

$$\lim_{h\to 0} \frac{f\left(v_0^k + h\,(v_i^k - v_0^k)\right) - f(v_0^k)}{h} \quad < \quad 0\,.$$

Thus, there exists an $h > 0$ such that

$$f\left(v_0^k + h\,(v_i^k - v_0^k)\right) \quad < \quad f(v_0^k)\,.$$

To see that the algorithm produces a point $v_0^k + h\,(v_t - v_0^k)$, $v_t \in \overline{v_0^k v_i^k}$, such that

$$f\left(v_0^k + h\,(v_t - v_0^k)\right) \;<\; f(v_0^k)\,,$$

consider the following: if a satisfactory step is not found, then the simplex contracts. So, for $i = 1, \ldots, n$, $v_i^{k+1} \leftarrow v_{c_i}^k$. The vertices of the contracted simplex are defined to be

$$v_{c_i}^k \;=\; v_0^k + \theta\,(v_i^k - v_0^k)\,,$$

for $i = 1, \ldots, n$, where $\theta \in (0,1)$ is the fixed contraction factor. This means that the sequence $\left\{v_i^k\right\}$ is converging toward v_0^k with constant θ. It then follows that there exists a p_k such that $\theta^{p_k} < h$. Therefore, for any k, there exists a positive integer p_k, such that

$$f(v_0^{k+p_k}) \;<\; f(v_0^k)\,.$$

$\square$

Lemma 3.1 leads us to Lemma 3.2.

Lemma 3.2 Suppose that (1) f is differentiable, and (2) $L(v_0^0)$ is compact. Then there is some $\hat{f}$ such that

$$\lim_{k\to\infty} f(v_0^k) \;=\; \hat{f}\,.$$

Furthermore, $\left\{v_0^k\right\}$ has at least one limit point $\hat{x}$, and

$$\lim_{k\to\infty}\left[\inf_{x\in C(\hat{x})}\left\|v_0^k - x\right\|\right] \;=\; 0\,.$$

Proof The proof of the first statement follows directly from Lemma 3.1: $\hat{f}$ exists because $\left\{f(v_0^k)\right\}$ is a monotone non-increasing sequence which is bounded below.

To prove the second statement we consider the following: Since $L(v_0^0)$ is compact, $\left\{v_0^k\right\}$ contains a convergent subsequence. We will denote this convergent subsequence as $\left\{v_0^{k_i}\right\}$ and say that $v_0^{k_i} \to \hat{x}$. By continuity, $f(\hat{x}) = \lim_{i\to\infty} f(v_0^{k_i}) = \hat{f}$.

Define

$$\delta_k \quad = \quad \inf_{x \in C(\hat{x})} \left\| v_0^k - x \right\| .$$

Note that since $L(v_0^0)$ is compact, δ_k is bounded; i.e., there exists a $B \geq 0$ such that $0 \leq \delta_k \leq B$. But $\lim_{i\to\infty} \delta_{k_i} = 0$, which implies that

$$\liminf_{k\to\infty} \delta_k \quad = \quad 0 .$$

Next, suppose that $\{\delta_{k_i}\}$ is *any* (infinite) convergent subsequence of $\{\delta_k\}$. There is a corresponding sequence of $\left\{v_0^{k_i}\right\}$; this sequence of best vertices has a convergent subsequence which we denote also by $\left\{v_0^{k_i}\right\} \to \check{x}$. Again, $f(\check{x}) = \lim_{i\to\infty} f(v_0^{k_i}) = \hat{f}$, so $\check{x} \in C(\hat{x})$. Thus, $\delta_{k_i} \to 0$. Hence, the only possible accumulation points of $\{\delta_k\}$ are 0 or ∞. But, $\delta^k \leq B$, so

$$\limsup_{k\to\infty} \delta_k \quad = \quad 0 .$$

Since $\liminf_{k\to\infty} \delta_k = \limsup_{k\to\infty} \delta_k = 0$, we have $\lim_{k\to\infty} \delta_k = 0$ as required.

$\square$

We have now established that the multi-directional search algorithm is a descent method with a strictly monotonically decreasing subsequence of the function values at the best vertices, that the sequence of function values converges, and that the sequence of best vertices converge to the corresponding level set.

3.2.2 Safeguarding the search directions

While the algorithm does generate at least one direction which produces descent, we still need to guarantee that the sequence of search directions does not become arbitrarily bad; i.e., that the sequence of search directions does not become arbitrarily close to being orthogonal to the gradient. To ensure this, it is enough to show that the search directions p^k are uniformly bounded away from becoming orthogonal to the gradient of f at the iterate x^k:

$$\frac{\left|\nabla f(x_k)^T p^k\right|}{\|\nabla f(x_k)\| \; \|p^k\|} \;\; \geq \;\; \gamma\,, \tag{3.6}$$

where γ is a strictly positive constant independent of k.

The multi-directional search algorithm gives us this condition for free. Since the simplex never changes its original shape, we are guaranteed that at least one search direction (edge) satisfies (3.6) at each iteration. To see this, we note that because the simplex never changes its original shape, the set of edges adjacent to the best vertex v_0^k remains *uniformly* linearly independent. This means that at each step k there exists a constant $\gamma > 0$ which does not depend on k, such that

$$\max\left\{\frac{|x^T(v_0^k - v_i^k)|}{\|x\|\,\|v_0^k - v_i^k\|}, i = 1, \ldots, n\right\} \;\; \geq \;\; \gamma \tag{3.7}$$

for all $x \in I\!R^n$, $x \neq 0$. Thus, if f is differentiable, at every iteration the angle between the gradient of f at v_0^k and at least one search direction is uniformly bounded away from $90°$.

3.2.3 Enforcing step length control

We have shown that the multi-directional search algorithm generates at least one direction of descent that is uniformly bounded away from becoming orthogonal to the gradient, but we still need to ensure that the steps the algorithm takes are neither too long nor too short. In most other descent methods, step length control is guaranteed by enforcing the Armijo-Goldstein-Wolfe conditions. To prevent steps that are too long, the Armijo-Goldstein-Wolfe conditions require the average rate of decrease in the function value from the current iterate to the next iterate be at least some prescribed fraction of the initial rate of decrease in that direction. Thus, the so called "alpha" condition is enforced:

$$f(x^{k+1}) \;\; \leq \;\; f(x^k) + \alpha \nabla f(x^k)^T (x^{k+1} - x^k)\,, \tag{3.8}$$

with $\alpha \in (0,1)$. To prevent steps that are too short, the Armijo-Goldstein-Wolfe conditions require the rate of decrease of the function f in the search direction at the new iterate be larger than some prescribed fraction of the rate of decrease in the search direction at the current iterate. This leads to the "beta" condition:

$$\nabla f(x^{k+1})^T (x^{k+1} - x^k) \;\; \geq \;\; \beta \nabla f(x^k)^T (x^{k+1} - x^k)\,, \tag{3.9}$$

with $\beta \in (\alpha, 1)$. For a further discussion of the Armijo-Goldstein-Wolfe conditions, see, for example, either Dennis and Schnabel [14], Gill, Murray, and Wright [16], or Ortega and Rheinboldt [25].

The multi-directional search algorithm cannot explicitly enforce the Armijo-Goldstein-Wolfe conditions because it does not have explicit knowledge of the gradient. However, there is enough structure in the algorithm to enforce step length control without enforcing the Armijo-Goldstein-Wolfe conditions.

The crux of the proof of Theorem 3.1 lies in showing that if we suppose that no limit point of $\left\{ v_0^k \right\}$ is a stationary point, then all but finitely many $\left\{ v_0^k \right\}$ are points where $\nabla f(v_0^k)$ must be bounded away from zero, independently of k. We will show that then the algorithm generates a finite number of points, which contradicts Lemma 3.1. It is this argument which is both the most interesting, and the most illuminating, part of the proof.

We begin by proving the existence of an upper bound on the lengths of the edges of the simplex.

Claim 3.1 Suppose that (1) f is differentiable and (2) $L(v_0^0)$ is compact. Then there exists a constant $M > 0$ such that

$$\left\| v_i^k - v_0^k \right\| \leq M \quad \forall\, i, k\,.$$

Proof Recall that $L(v_0^0)$ is defined to be

$$L(v_0^0) = \left\{ x : f(x) \leq f(v_0^0) \right\}.$$

Assume that v_0^0 is not a stationary point. If so, Lemma 3.1 guarantees that the multi-directional search algorithm will produce a new best vertex in a finite number of iterations. We will denote by k_1 the iteration which first produced a new best vertex. We have also assumed that $L(v_0^0)$ is compact. Thus, for all $k > k_1$, there exists at least one vertex v_i^k such that the vertices $v_0^k, v_i^k \in L(v_0^0)$. Since $L(v_0^0)$ is bounded, this implies a bound on the length of the edge $\overline{v_0^k v_i^k}$. Since the rescaling factors are constant across all edges for all iterations, the relative lengths of all edges in the simplex remain the same across all iterations of the algorithm. This implies the existence of $M > 0$ as required.

If v_0^0 is a stationary point, then we cannot apply Lemma 3.1. Thus we cannot guarantee that in a finite number of iterations we will find a new best vertex. There are then two possibilities: The first possibility is that the multi-directional search algorithm does produce a new best vertex — consider, for instance, the case where v_0^0 is at a local maximizer. If so, the argument given above still holds. The other possibility is that the multi-directional search does not find a new best vertex, i.e., $v_0^k = v_0^0$ for every k. If so, then only the contraction step has been taken because the reflection step and the expansion step are accepted only when they do produce a new best vertex. Since the contraction factor θ is strictly less than one, this means that the length of every edge of the simplex is strictly monotonically decreasing. Thus, the maximum length across all edges in the initial simplex provides an upper bound on the length of all edges in all subsequent simplices, as required.

□

The existence of an upper bound on the lengths of the edges of the simplex implies the existence of a compact set, which we will call $\mathcal{M}$, that contains $L(v_0^0)$ and all the simplices generated by the multi-directional search algorithm.

Next we will show that if the sequence of best vertices stays bounded away from stationary points, then there is a lower bound on the lengths of the edges in the simplices the algorithm generates. To show that under the above hypothesis a lower bound on the lengths of the edges in the simplex does exist, we will show that once the edges in the simplex become small enough, the reflection step will always be acceptable. In that case, while the expansion step will be considered, the contraction step will not. Since the lengths of the edges in the simplex are reduced only when the simplex is contracted, this argument ensures no further reduction in the size of the simplex.

Claim 3.2 Suppose (1) f is continuously differentiable and (2) $L(v_0^0)$ is compact. Assume that for all $k \geq k_0$ there exists a constant $\sigma > 0$, which does not depend on k, such that

$$\left\| \nabla f(v_0^k) \right\| \;\; \geq \;\; \sigma \, .$$

Then there exists a constant $m > 0$ such that

$$m \;\; \leq \;\; \left\| v_i^k - v_0^k \right\| \quad \forall \; i, k \, .$$

Proof Let $\mathcal{M}$ be the compact set of Claim 3.1. By hypothesis, $\nabla f(x)$ is a continuous function on the compact set $\mathcal{M}$. This means that $\nabla f(x)$ is uniformly continuous on $\mathcal{M}$.

The uniform linear independence of the set of search directions at each iteration gives us the constant $\gamma > 0$, while the hypothesis that the gradient is bounded away from zero gives us the constant $\sigma > 0$. By the uniform continuity of $\nabla f(x)$ there exists a $\delta > 0$, depending only on σ and γ, such that for all k,

$$\left\|\nabla f(x) - \nabla f(v_0^k)\right\| \;<\; \sigma\gamma/2 \quad \text{whenever} \quad \left\|x - v_0^k\right\| \;<\; \delta\,.$$

Now we will show that if at any iteration k we choose a vertex v_i^k, $i \neq 0$, which satisfies

$$\frac{\left|\nabla f(v_0^k)^T(v_i^k - v_0^k)\right|}{\left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|} \;\geq\; \gamma\,,$$

then once the edges of the simplex become "small enough" the simplex will not contract, so it cannot get any smaller.

Again, we have two cases to consider:

- Case 1: $\nabla f(v_0^k)^T(v_i^k - v_0^k) > 0$ [See Figure 3.2.]
- Case 2: $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$ [See Figure 3.3.]

Case 1: $\nabla f(v_0^k)^T(v_i^k - v_0^k) > 0$

By definition

$$v_{r_i}^k \;=\; v_0^k + (v_0^k - v_i^k) \qquad \text{so} \qquad v_i^k - v_0^k \;=\; -(v_{r_i}^k - v_0^k)\,.$$

We invoke the Mean Value Theorem to get

$$f(v_{r_i}^k) - f(v_0^k) \;=\; \nabla f(\xi)^T(v_{r_i}^k - v_0^k)$$

where $\xi \in \overline{v_0^k v_{r_i}^k}$.

Add and subtract $\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)$ to obtain:

$$\begin{aligned} f(v_{r_i}^k) - f(v_0^k) &= \nabla f(v_0^k)^T(v_{r_i}^k - v_0^k) \\ &+ \left(\nabla f(\xi) - \nabla f(v_0^k)\right)^T (v_{r_i}^k - v_0^k). \end{aligned} \tag{3.10}$$

Consider the first term on the right hand side of (3.10), $\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)$. We chose v_i^k so that

$$\frac{\left|\nabla f(v_0^k)^T(v_i^k - v_0^k)\right|}{\left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|} \geq \gamma$$

or

$$\left|\nabla f(v_0^k)^T(v_i^k - v_0^k)\right| \geq \gamma \left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|.$$

Since $\nabla f(v_0^k)^T(v_i^k - v_0^k) > 0$ we then have

$$\nabla f(v_0^k)^T(v_i^k - v_0^k) \geq \gamma \left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|.$$

By construction, $\left\|v_{r_i}^k - v_0^k\right\| = \left\|v_i^k - v_0^k\right\|$. Thus,

$$\nabla f(v_0^k)^T(v_i^k - v_0^k) \geq \gamma \left\|\nabla f(v_0^k)\right\| \left\|v_{r_i}^k - v_0^k\right\|.$$

Since $v_i^k - v_0^k = -(v_{r_i}^k - v_0^k)$, we get

$$\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k) \leq -\gamma \left\|\nabla f(v_0^k)\right\| \left\|v_{r_i}^k - v_0^k\right\|. \tag{3.11}$$

Now, consider the second term on the right hand side of (3.10). The Cauchy-Schwartz inequality gives us:

$$\left|\left(\nabla f(\xi) - \nabla f(v_0^k)\right)^T (v_{r_i}^k - v_0^k)\right| \leq \left\|\nabla f(\xi) - \nabla f(v_0^k)\right\| \left\|v_{r_i}^k - v_0^k\right\|. \tag{3.12}$$

We combine (3.11) and (3.12) to rewrite (3.10) as

$$f(v_{r_i}^k) - f(v_0^k) \leq -\gamma \left\|\nabla f(v_0^k)\right\| \left\|v_{r_i}^k - v_0^k\right\| + \left\|\nabla f(\xi) - \nabla f(v_0^k)\right\| \left\|v_{r_i}^k - v_0^k\right\|.$$

Observe that since ∇f is uniformly continuous, there exists a $\delta > 0$, depending only on σ and γ, such that for all k

$$\left\|\nabla f(x) - \nabla f(v_0^k)\right\| \ < \ \sigma\gamma/2 \quad \text{whenever} \quad \left\|x - v_0^k\right\| \ < \ \delta\,.$$

We then have

$$f(v_{r_i}^k) - f(v_0^k) \ \leq \ \underbrace{\underbrace{(\underbrace{-\gamma \underbrace{\left\|\nabla f(v_0^k\right\|}_{\geq\sigma}}_{\leq-\sigma\gamma} + \underbrace{\left\|\nabla f(v_{r_i}^k) - \nabla f(v_0^k)\right\|}_{<\frac{\sigma\gamma}{2}})}_{<-\frac{\sigma\gamma}{2}<0} \underbrace{\left\|v_{r_i}^k - v_0^k\right\|}_{>0}}_{<0}.$$

Thus,

$$f(v_{r_i}^k) - f(v_0^k) \ < \ 0 \ \Longrightarrow \ f(v_{r_i}^k) \ < \ f(v_0^k)\,,$$

whenever $\left\|\xi - v_0^k\right\| < \delta$. Therefore, the simplex will not contract.

$$\boxed{\textbf{Conclusion:} \ \ v_{r_i}^k \ \textit{is acceptable}}$$

Remark

We can actually say more. We have shown that

$$f(v_{r_i}^k) - f(v_0^k) \ < \ -\frac{\sigma\gamma}{2}\left\|v_{r_i}^k - v_0^k\right\|$$

or

$$f(v_{r_i}^k) \ < \ f(v_0^k) - \frac{\sigma\gamma}{2}\left\|v_{r_i}^k - v_0^k\right\|.$$

Recall we have assumed that for all k

$$\sigma \ \leq \ \left\|\nabla f(v_0^k)\right\|$$

so that

$$f(v_{r_i}^k) \;<\; f(v_0^k) - \frac{\gamma}{2}\left\|\nabla f(v_0^k)\right\|\left\|v_{r_i}^k - v_0^k\right\|. \tag{3.13}$$

Again, we call upon the Cauchy-Schwartz inequality to obtain

$$\left|\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)\right| \;\leq\; \left\|\nabla f(v_0^k)\right\|\left\|v_{r_i}^k - v_0^k\right\|$$

so that

$$-\frac{\gamma}{2}\left|\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)\right| \;\geq\; -\frac{\gamma}{2}\left\|\nabla f(v_0^k)\right\|\left\|v_{r_i}^k - v_0^k\right\|. \tag{3.14}$$

Substituting (3.14) into (3.13) gives us

$$f(v_{r_i}^k) \;<\; f(v_0^k) - \frac{\gamma}{2}\left|\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)\right|. \tag{3.15}$$

However, we have assumed $\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k) < 0$, so

$$-\frac{\gamma}{2}\left|\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k)\right| \;=\; \frac{\gamma}{2}\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k). \tag{3.16}$$

Substituting (3.16) into (3.15) we then have

$$f(v_{r_i}^k) \;<\; f(v_0^k) + \frac{\gamma}{2}\nabla f(v_0^k)^T(v_{r_i}^k - v_0^k). \tag{3.17}$$

In other words, when the lengths of the edges in the simplex become "small enough," we are guaranteed a uniform fraction of Cauchy decrease — a fraction that depends on γ, which is the constant from the uniform linear independence condition. Thus, when the lengths of the edges in the simplex become "small enough," (3.17) shows that we satisfy the alpha condition of Armijo-Goldstein-Wolfe for Case 1.

One point worth noting is that we are only guaranteed to satisfy the alpha condition when $\|\nabla f(v_0^k)\| > \sigma$ and the simplex becomes "small enough." In general, while we are guaranteed simple decrease, there is no guarantee that the alpha condition is, in fact, satisfied. In some iterations, we may do substantially better; in others we may not do as well. Rather, this observation suggests that in the "worst" case we will at least see a uniform fraction of Cauchy decrease.

Case 2: $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$

Our argument for Case 2 follows that for Case 1, but leads to a very different conclusion.

We again invoke the Mean Value Theorem to obtain

$$f(v_i^k) - f(v_0^k) \;\; = \;\; \nabla f(\xi)^T(v_i^k - v_0^k)$$

where $\xi \in \overline{v_0^k v_i^k}$. We add and subtract $\nabla f(v_0^k)^T(v_i^k - v_0^k)$ to obtain

$$\begin{aligned} f(v_i^k) - f(v_0^k) \;\; &= \;\; \nabla f(v_0^k)^T(v_i^k - v_0^k) \\ &\quad + \left(\nabla f(\xi) - \nabla f(v_0^k)\right)^T (v_i^k - v_0^k). \end{aligned} \tag{3.18}$$

Consider the first term on the right hand side of (3.18), $\nabla f(v_0^k)^T(v_i^k - v_0^k)$. We chose v_i^k so that

$$\frac{\left|\nabla f(v_0^k)^T(v_i^k - v_0^k)\right|}{\left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|} \;\; \geq \;\; \gamma\,.$$

Thus,

$$\left|f(v_0^k)^T(v_i^k - v_0^k)\right| \;\; \geq \;\; \gamma \left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|.$$

However, we know that $\nabla f(v_0^k)^T(v_i^k - v_0^k) < 0$, so that

$$f(v_0^k)^T(v_i^k - v_0^k) \;\; \leq \;\; -\gamma \left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|. \tag{3.19}$$

Now, consider the second term on the right hand side of (3.18). The Cauchy-Schwartz inequality gives us

$$\left|(\nabla f(\xi) - \nabla f(v_0^k))^T(v_i^k - v_0^k)\right| \;\; \leq \;\; \left\|\nabla f(\xi) - \nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|. \tag{3.20}$$

We combine (3.19) and (3.20) to rewrite (3.18) as

$$f(v_i^k) - f(v_0^k) \;\; \leq \;\; -\gamma \left\|\nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\| + \left\|\nabla f(\xi) - \nabla f(v_0^k)\right\| \left\|v_i^k - v_0^k\right\|.$$

Now observe that since ∇f is uniformly continuous, there exists a $\delta > 0$, depending only on σ and γ, such that for all k

$$\left\|\nabla f(x) - \nabla f(v_0^k)\right\| \;<\; \sigma\gamma/2 \quad \text{whenever} \quad \left\|x - v_0^k\right\| \;<\; \delta\,.$$

We then have

$$f(v_i^k) - f(v_0^k) \;\leq\; \underbrace{\underbrace{\underbrace{(-\gamma \underbrace{\left\|\nabla f(v_0^k)\right\|}_{\geq\sigma}}_{\leq -\sigma\gamma} + \underbrace{\left\|\nabla f(\xi) - \nabla f(v_0^k)\right\|}_{<\frac{\sigma\gamma}{2}})}_{<-\frac{\sigma\gamma}{2}<0} \underbrace{\left\|v_i^k - v_0^k\right\|}_{>0}}_{<0}\,.$$

Thus

$$f(v_i^k) - f(v_0^k) \;<\; 0 \;\Longrightarrow\; f(v_i^k) \;<\; f(v_0^k)\,.$$

whenever $\left\|\xi - v_0^k\right\| < \delta$.

But this leads to a contradiction! We have chosen v_i^k so that $i \neq 0$ and the algorithm requires that

$$f(v_0^k) \;\leq\; f(v_i^k) \qquad \forall\; i = 1, \ldots, n\,.$$

Conclusion: Case 2 cannot happen.

We have now shown that if $\left\|\xi - v_0^k\right\| < \delta$ then for Case 1 the algorithm will accept the reflection step and the simplex will not contract and Case 2 cannot occur. In Case 1,

$$\xi \;\in\; \overline{v_0^k v_{r_i}^k}$$

while in Case 2

$$\xi \;\in\; \overline{v_0^k v_i^k}\,.$$

By construction, $\left\|v_{r_i}^k - v_0^k\right\| = \left\|v_i^k - v_0^k\right\|$, so that

$$\left\|\xi - v_0^k\right\| \;<\; \left\|v_i^k - v_0^k\right\|.$$

This implies the existence of a constant $m > 0$ such that

$$m \;\leq\; \left\|v_i^k - v_0^k\right\| \quad \forall\, i, k\,,$$

if for all k there exists a constant $\sigma > 0$, which does not depend on k, such that

$$\left\|\nabla f(v_0^k)\right\| \;\geq\; \sigma\,.$$

□

Now that we have established lower and upper bounds for the lengths of the edges in the simplex, we are ready to show that given these lower and upper bounds, the multi-directional search algorithm can only visit a finite number of points.

To see how this argument works, we begin by considering how the algorithm decides which points to visit. In Figure 3.4 we assume that we are given an initial simplex and that we know which vertex in that simplex is "best." The algorithm automatically visits the reflected simplex. The result of the acceptance test then dictates whether the expanded or the contracted simplex will be visited. In any event, given an initial simplex, and the best vertex in that simplex, we can list, in advance, all the simplices that can be generated at the current iteration. This means we can enumerate, *a priori*, all the points that can possibly be visited during the current iteration.

Now assume that we are given an initial simplex, but do not know any information about the function values at any of the vertices in the simplex. This means that we have no way of identifying the “best” vertex in the simplex. Even without this information, we can still list all the simplices that might be generated at the current iteration and we can still enumerate all the points that might be visited during the iteration. To do this, we simply consider the possibilities when each of the vertices in the original simplex is allowed to be “best,” as shown in Figure 3.5.

We now extend this speculation to the next iteration. Again, we assume no knowledge of the function value at any of the vertices. We allow each vertex in each of the trial simplices generated at the previous iteration to be “best.” For our example, seen in Figure 3.6, we do begin to enforce a lower bound on the lengths of the edges in any of the simplices. Furthermore we require our simplices to be contained in a compact set. Both of these restrictions are important because they eliminate several of the simplices that might otherwise have been considered.

Consider yet another iteration. Again we allow each vertex in each of the trial simplices generated at the previous iteration to be “best,” but again we apply our restrictions to eliminate even more possible simplices. The result can be seen in Figure 3.7.

Finally, if we remove all the edges, we see in Figure 3.8 that the method is, in fact, generating a grid. Since this grid must be contained in a compact set, and since its mesh size is fixed, this means that there are only a finite number of points which the algorithm can visit. We required the lower bound on the lengths of the edges in the simplex to fix the mesh size of the grid. We required the upper bound on the lengths of the edges in the simplex to give us a compact set over which to search. However, once we accept these two restrictions, this means we can predict — without knowledge of any function information — all the points the algorithm can visit from any initial simplex.

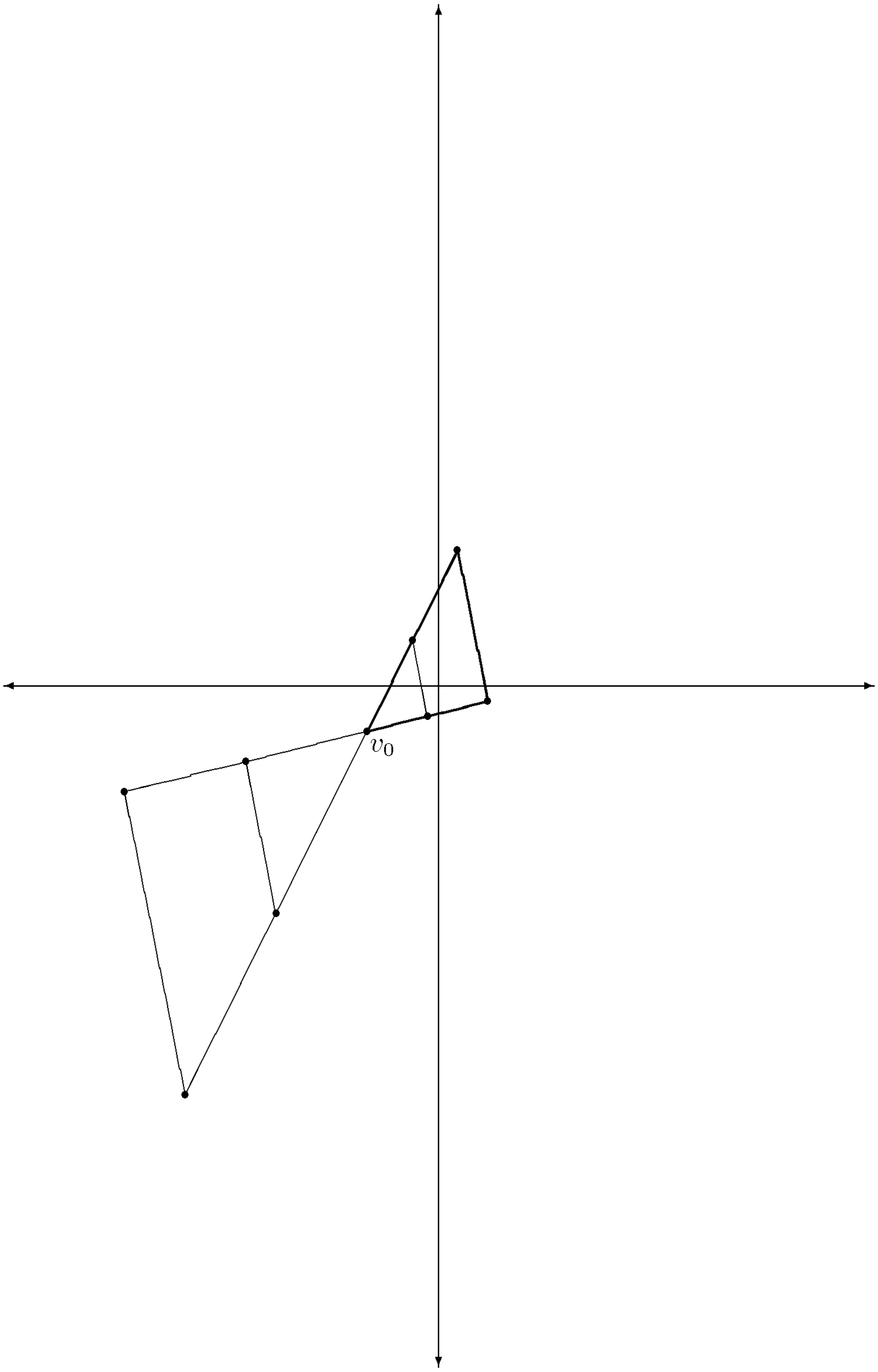

Figure 3.4 Enumerating the vertices — when we know the best vertex

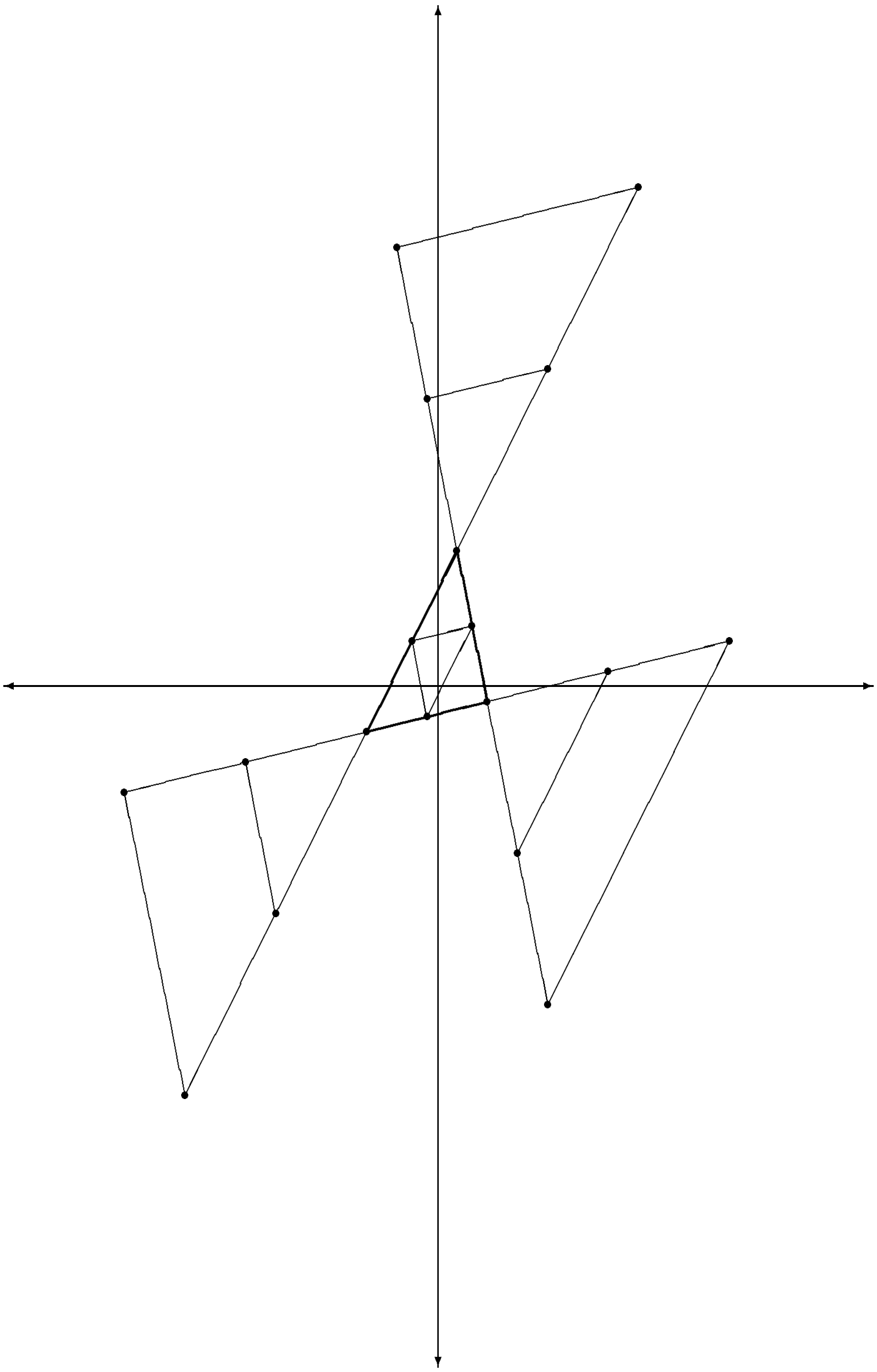

Figure 3.5 Enumerating the vertices —
when we do not know the best vertex

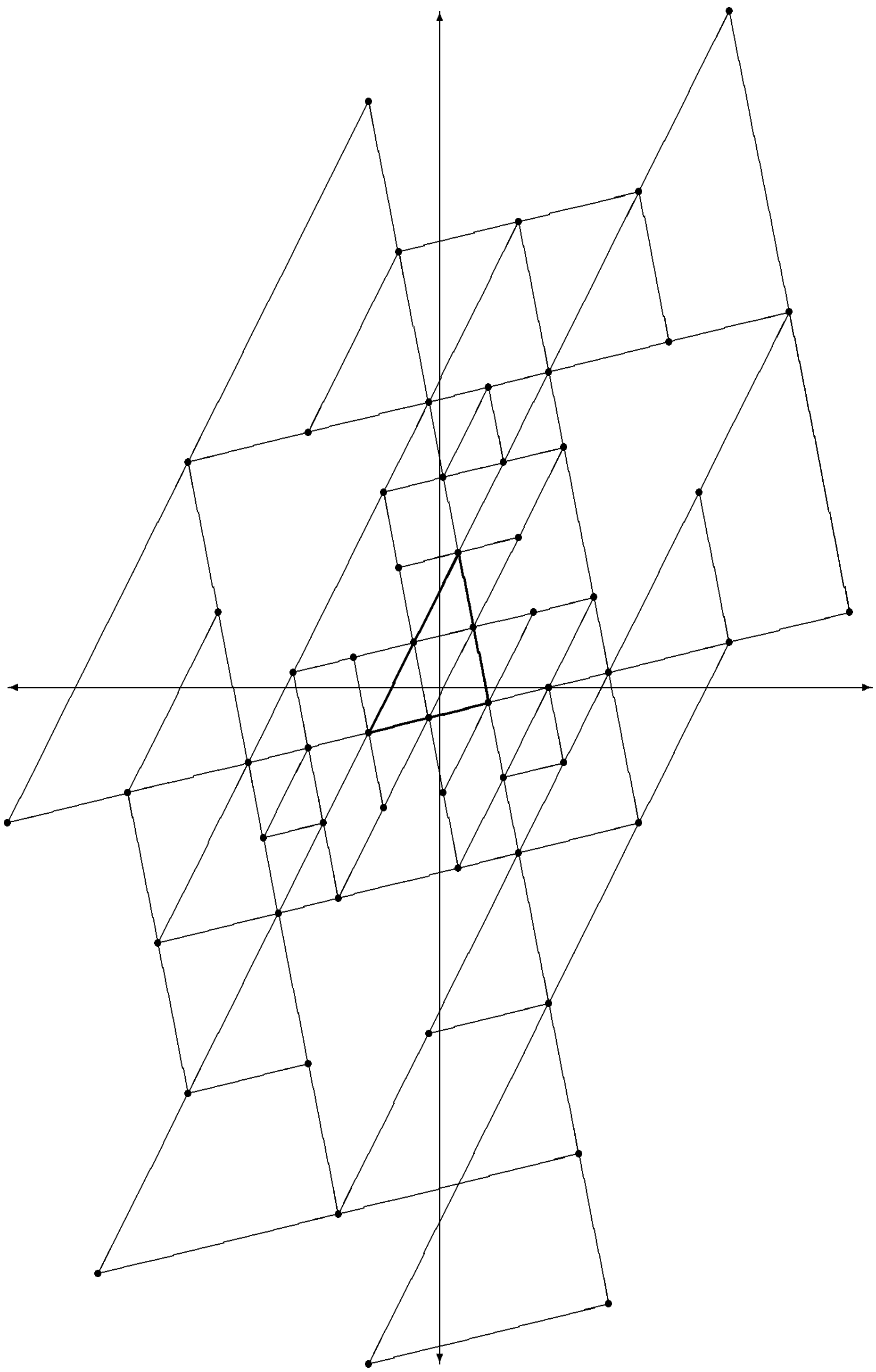

Figure 3.6 Enumerating the vertices — after one additional iteration

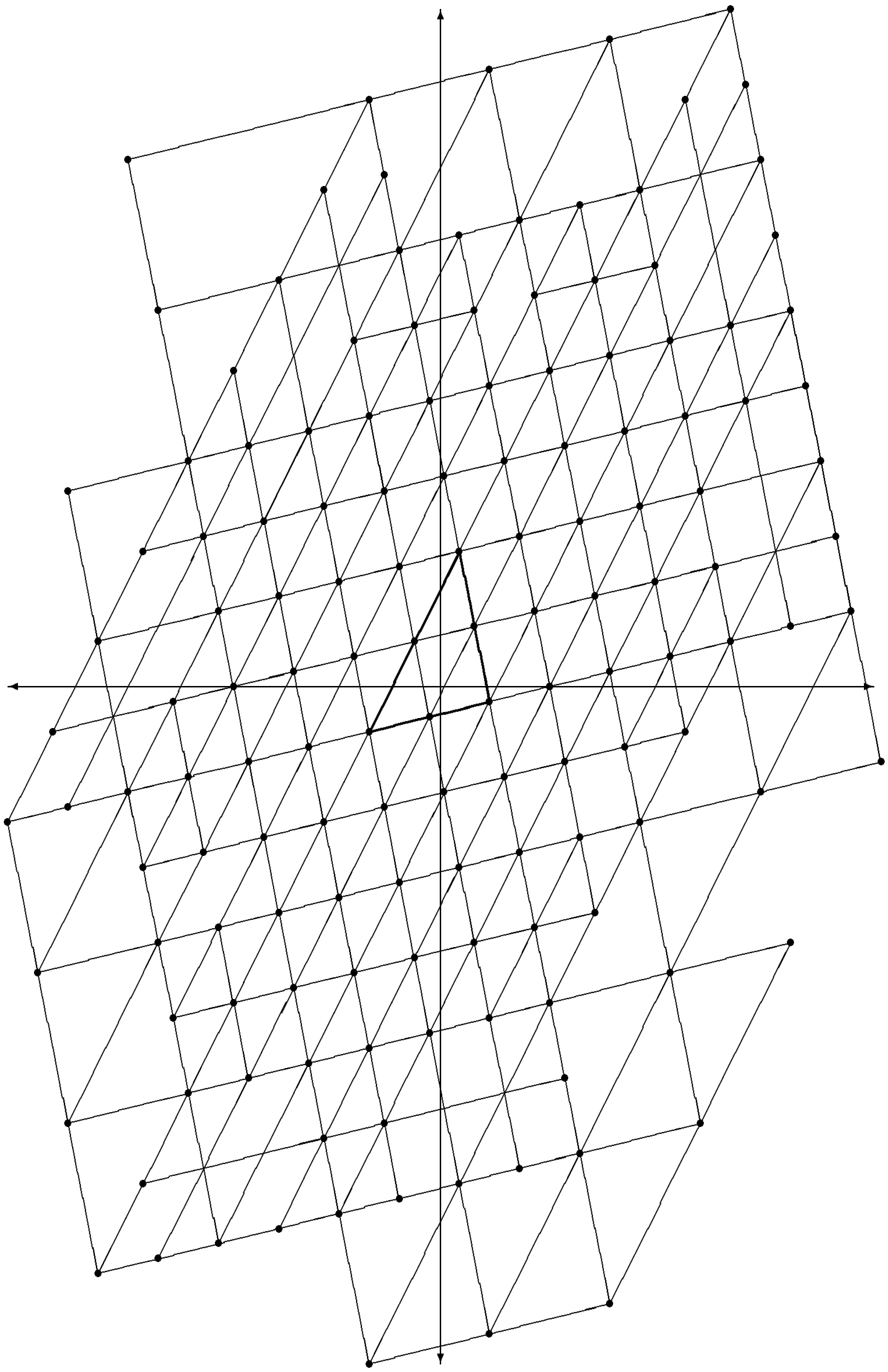

Figure 3.7 Enumerating the vertices — after two additional iterations

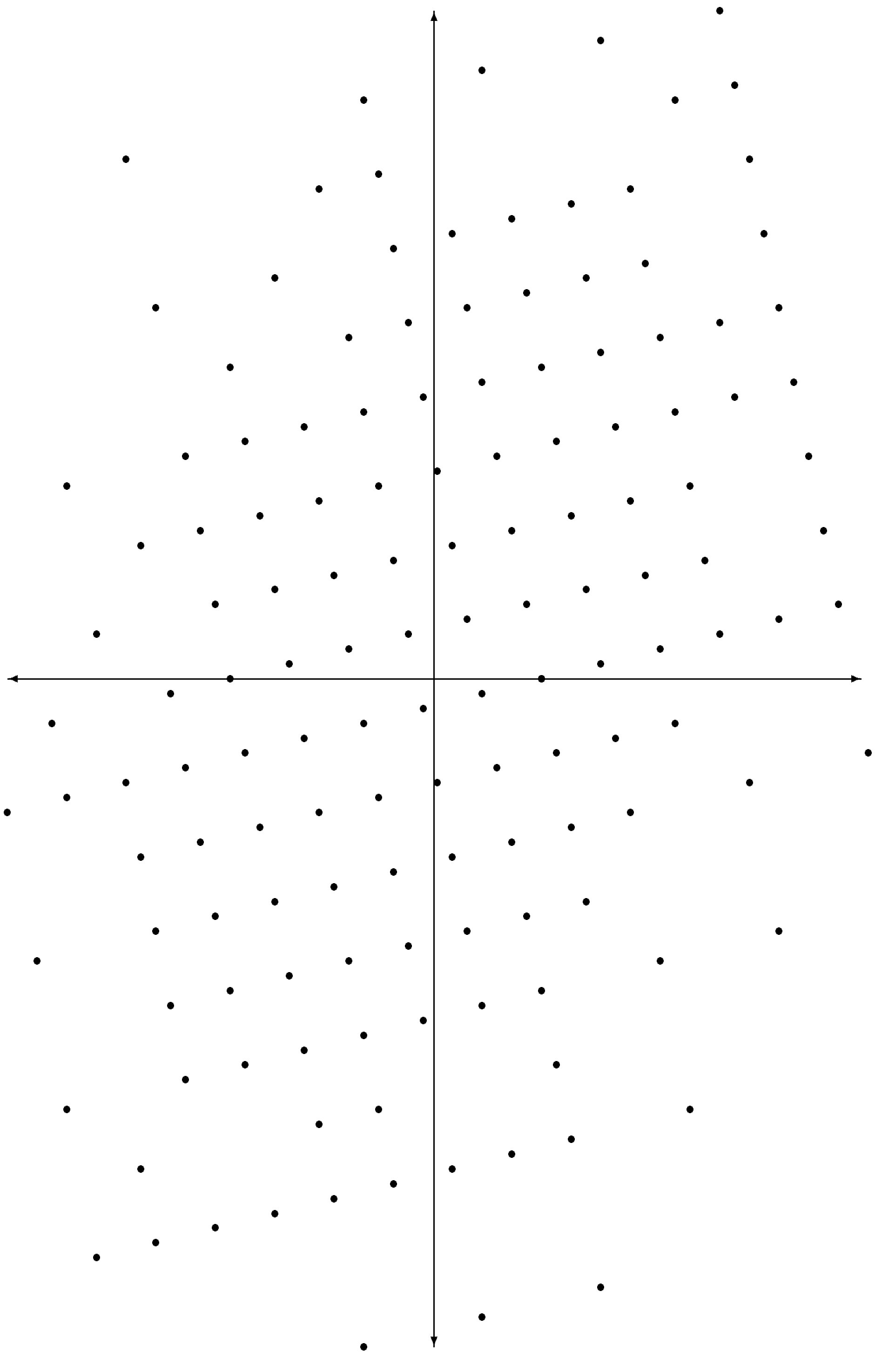

Figure 3.8 Enumerating the vertices — after removing all the edges

Now we will prove that given any initial simplex, if we assume that the norm of the gradient at the best vertex is uniformly bounded away from zero by a constant σ, the multi-directional search algorithm can only visit finite number of points.

Claim 3.3 Suppose that (1) f is continuously differentiable and (2) $L(v_0^0)$ is compact. Assume that for all k there exists a constant $\sigma > 0$, which does not depend on k, such that

$$\left\|\nabla f(v_0^k)\right\| \;\geq\; \sigma\,.$$

Then the multi-directional search algorithm can only visit a finite number of points.

Proof The proof is by construction.

Take the initial simplex. Designate any one of its vertices as "best." (There is no need to consider function information at this point.)

Rescale the simplex, using the contraction constant θ, until every edge in the simplex satisfies the condition

$$m \;\leq\; \|v_i - v_0\| \quad \leq \delta\,,$$

for all $i = 1, \ldots, n$.

Find the least common denominator for the scale factors θ and μ. This makes sense since we have stipulated that θ and μ must be rational numbers.

Divide the least common denominator by the reduction factor θ. Reduce the simplex one last time by this factor.

Take the set of edges adjacent to the best vertex

$$\{(v_i - v_0), i = 1, \ldots, n\}$$

as a basis for the grid. Now take all *integer* multiples of the basis that generate points inside the compact set $\mathcal{M}$.

This will give a grid with fixed mesh size inside a compact set. Therefore the number of points in the grid will be finite.

Furthermore, every possible simplex, given the initial simplex, will map onto this grid since all possible step sizes are integer multiples of the mesh size.

Therefore, the algorithm can only visit a finite number of points. □

3.3 Proof of the convergence theorem

Now that we have established that the multi-directional search algorithm is a descent method, and that we can guarantee first, that the search directions will not deteriorate, and second, that the steps taken by the algorithm cannot become too long or too short, we are ready to prove Theorem 3.1.

Proof The proof is by contradiction.

Suppose for all but finitely many k there exists a constant $\sigma > 0$, which does not depend on k, such that

$$\left\|\nabla f(v_0^k)\right\| \geq \sigma .$$

Then, taken together, the upper and lower bounds on the length of the edges in the simplex (Claim 3.1 and Claim 3.2), imply that the algorithm can only visit a finite number of points (Claim 3.3). But this contradicts Lemma 3.1, which guarantees us *strict* decrease on the function value at the best vertex in a finite number of iterations. Thus, the hypothesis cannot hold, which means that

$$\liminf_{k\to\infty} \left\|\nabla f(v_0^k)\right\| = 0 .$$

Then there exists some subsequence of the best vertices, $\left\{v_0^k\right\}$, which converges to a point $x_* \in X_*$.

We invoke Lemma 3.2 to complete the proof.

□

Chapter 4

Implementation Details

In Chapter 2 we gave both a general description and a formal statement of the multi-directional search algorithm. However, before we can address the performance of the algorithm, there are several implementation details that remain to be discussed. In particular:

- How do we choose an initial simplex?
- How do we choose the expansion and contraction parameters μ and θ?
- How do we decide when to stop the algorithm?

The convergence analysis does provide us with some guidelines on these issues, but most of our decisions were based on practical experience. Here we relied not only on our own experience with the multi-directional search algorithm, but also on the collective wisdom of others, acquired from years of experience with direct search methods — in particular, with the Nelder-Mead simplex algorithm. Armed with both these guidelines and our experience, we will now comment on the implementation of the algorithm.

Both the rationale for our particular choices, as well as a discussion of the alternatives, can be found in the next three sections. Our choices — used to test for the performance results given in Chapter 5 — can be found in Section 4.4.

4.1 Choosing an initial simplex

The multi-directional search algorithm requires only that the simplex used to start the procedure be nondegenerate; i.e., the set of $n+1$ points which defines the simplex must span $I\!R^n$. The reason for this restriction is clear: if the simplex is degenerate, the algorithm can only minimize over the subspace spanned by the degenerate simplex. However, choosing the shape, size, and orientation of the initial simplex is another matter entirely. Few criteria for the choice of these values have been published in the optimization literature.

4.1.1 Shape

The convergence theorem for the multi-directional search algorithm does provide some help in choosing the shape of the initial simplex. To see this, we return to the observation, made in Section 3.2.2, that at any vertex in the simplex, the set of edges adjacent to that vertex is uniformly linearly independent. Thus, at each step k there exists a constant $\gamma > 0$ which does not depend on k, such that

$$\max \left\{ \frac{|x^T(v_0^k - v_i^k)|}{\|x\| \, \|v_0^k - v_i^k\|}, i = 1, \ldots, n \right\} \quad \geq \quad \gamma$$

for all $x \in I\!R^n$, $x \neq 0$. How might we choose a simplex to obtain the "optimal" value of the lower bound γ? We quantify this question as follows: Given a simplex S, define Γ to be

$$\Gamma(S, x) \quad = \quad \min_{n+1 \text{vertices } v} \left(\max_{e = \text{edges adjacent to vertex } v} \frac{|e^T x|}{\|e\| \, \|x\|} \right).$$

We are then interested in choosing a simplex which solves the problem

$$\text{maximize} \qquad \min_{\|x\|=1} \Gamma(S, x). \tag{4.1}$$

A regular simplex, i.e., one in which all the edges are the same length, is a solution to (4.1). Since $\nabla f(v_0^k)$ could lie in any direction it would thus seem that, in general, we should start the multi-directional search algorithm with a regular simplex.

The use of a regular simplex is also consistent with much of the literature concerning the other two direct search simplex algorithms, the Nelder-Mead simplex algorithm [24] and the simplex algorithm of Spendley, Hext, and Himsworth [32]. In fact, Spendley, Hext, and Himsworth specify that their algorithm start with a regular simplex. It is also interesting to note that in their method the shape of the simplex remains fixed across all iterations of the algorithm.

Nelder and Mead place no restrictions, other than nondegeneracy, on the shape of the initial simplex. Most sources simply note this mild restriction but do not suggest ways to generate an initial simplex from a given initial estimate (i.e., a single starting point). Those that do address this issue give varying suggestions. For instance, Jacoby, Kowalik and Pizzo [18] note that the Nelder-Mead simplex algorithm only requires a general simplex, but they suggest that the construction of a regular initial simplex assures that its vertices span the full space. (In fact, we currently use the simple procedure given in their book to generate an initial simplex from a given starting point.) Parkinson and Hutchinson [26] suggest starting the Nelder-Mead simplex algorithm with a right-angled simplex. This simplex can be generated by defining each of the n new vertices to be some fixed distance in each of the n coordinate directions from the initial guess. In other words, given the unit basis vectors e_i and scalars α_i, we take the initial guess x_0 and construct the n points $x_i = x_0 + \alpha_i\, e_i$. Again, we are guaranteed that this simplex will span the full space as long as the α_i's are nonzero. Parkinson and Hutchinson go on to note, however, that in their investigation into the efficiency of variants on the Nelder-Mead simplex algorithm, the shape of the initial simplex proved to be relatively unimportant. Their conjecture, which our own experience with the Nelder-Mead simplex algorithm confirms, is that this result is to be expected since the initial simplex is rapidly modified by the action of the algorithm.

We close by noting that the shape of the simplex is scale dependent, so that a regular simplex may not be a desirable choice if the variables differ widely in scale.

In this instance it may be preferable to construct a right-angled simplex as suggested above: from the initial estimate take a step in each of the coordinate directions e_i of length α_i, where α_i is an estimate of the relative scale of the x_i variable. The advantage of the multi-directional search algorithm is that any information about the relative scale of each of the variables that is used to select the α_i's will be preserved across all iterations of the algorithm. The same cannot be said for the Nelder-Mead simplex algorithm. However, in the absence of any particular knowledge about the problem to be solved, it is perhaps "safest" to start the multi-directional search algorithm with a regular simplex — which is exactly what we choose to do.

4.1.2 Size

While the size of the initial simplex definitely affects performance, the scale dependency of the simplex makes general guidelines as to the best size for the initial simplex all but impossible. Since the problems we tested were not badly scaled, we typically chose a regular simplex with sides of length one. Thus, our experience on this issue is limited. The advantage of the multi-directional search algorithm is that the expansion and contraction steps automatically adjust the size of the entire simplex by rescaling the lengths of all the edges in the simplex. Thus, if the initial simplex is either too small or too large, the algorithm will rescale accordingly. The problem, particularly if the simplex is too large, is that the algorithm may spend a significant number of iterations simply contracting the simplex before it can make any real progress, and each iteration spent contracting the simplex requires $2n$ function evaluations.

For the Nelder-Mead simplex algorithm, the effect of size is much less clear. The Nelder-Mead simplex algorithm only rescales the entire simplex as a last resort. In this case it takes a "shrink" step — which is equivalent to the contraction step in the multi-directional search algorithm. Our experience shows that, in fact, the shrink step is almost never taken. (This will be discussed further in Chapter 5.) Instead, the Nelder-Mead simplex algorithm changes the size of the simplex by moving only one vertex at a time; however, each change also distorts the shape of the initial simplex. After n iterations the simplex may be smaller or larger, but its shape may also be significantly altered in the process. Our tests using the Nelder-Mead simplex algorithm would indicate that this interaction between changing the size of the simplex — and thus the shape — may prove disastrous. Further discussion of this issue will be deferred until Chapter 5.

Parkinson and Hutchinson [26] experimented with changing the size of the initial simplex used to start the Nelder-Mead simplex algorithm. They concluded that varying the size of the initial simplex did produce significant variations in the number of function evaluations required to solve each problem. Consequently, they suggested two strategies for determining the initial size of the simplex. Both strategies require line searches to determine the lengths of the edges in the simplex. Parkinson and Hutchinson reported modest decreases in the number of function evaluations required to minimize the function when either of these strategies were used to determine the size of the initial simplex. The trade-off is that more work, in the form of local line searches, must be done before starting the algorithm.

We make the following observations about the size of the simplex in the multi-directional search algorithm: As we have already noted, the multi-directional search algorithm automatically adjusts the size of the entire simplex by rescaling the lengths of all the edges in the simplex. The problem, particularly when the simplex is too large, is that the algorithm is very conservative in its rescaling; it only halves or doubles the lengths of the edges — independent of any function information acquired during the course of the reflection step. Each rescaling requires the computation of n new function values, which makes the process costly. If we could accelerate the rescaling, we might see a significant decrease in the total number of function evaluations required to minimize the function.

One idea we have considered, but not yet implemented, addresses the issue of repeated contractions. Once we have computed the reflection step we have enough information to construct an approximating quadratic along each of the n edges adjacent to the best vertex. It would then be possible to predict, at nominal additional cost, an approximate minimizer along each of those directions. We do not want the simplex to become any smaller than necessary, so we choose that minimizer which requires the least reduction in the size of the simplex. Now, we choose a strictly positive power of the contraction factor that will reduce the simplex to a size that is close to that required to identify the predicted minimizer. The effect of this acceleration is the same as repeatedly contracting the simplex, however, it eliminates all the intervening function evaluations. Ideally, the new simplex will produce a new best vertex while avoiding all the intermediate computations. The other advantage of this approach is that it preserves the fixed step sizes of the algorithm while making some attempt to better approximate the minimum along at least one of the edges.

The same approach could be used to accelerate the expansion step, but it is less clear that this will significantly improve the performance of the multi-directional search algorithm. In the first place, our experience indicates that the algorithm rarely takes more than two expansion steps in a row. Thus it would seem that the extra work, however minimal, would be difficult to justify. More importantly, recall that if a new best vertex is accepted — which would be the case if the expanded simplex were accepted — then at the next iteration the one direction we keep is the direction which produced the new best vertex (i.e., the edge $\overline{v_0^{k-1} v_0^k}$). Thus we continue to search along a direction for which we have already seen descent. In the meantime, however, we consider $n-1$ new search directions, one of which may produce even bigger decreases in the function value at the best vertex for the same relative step size. Thus we are less likely to pursue ever larger steps in a single direction at the expense of better relative decrease in a new direction.

In summary, there are no automatic procedures for choosing the initial size of the simplex. It should be clear that the multi-directional search algorithm can certainly recover from a bad initial guess, but such adjustments — particularly if the simplex is too large — may prove to be quite costly. We have suggested one possibility for overcoming this problem, but this idea has not yet been tested. We close by noting that as in the choice of shape, any information known about the function can, and should, be used to determine the size of the initial simplex.

4.1.3 Orientation

There is no question but that the orientation of the simplex will affect the progress of both the Nelder-Mead simplex algorithm and the multi-directional search algorithm. The reason is simple: the orientation determines the search directions. Parkinson and Hutchinson [26] found that varying the orientation of the initial simplex had a dramatic effect for all the functions they tested using the Nelder-Mead simplex algorithm. They restricted their attention to problems with only two variables and then rotated the simplex about the best vertex in increments of $1°$. They found that in some cases a rotation of only $1°$ changed the number of function evaluations by $\pm 45\%$. Their conclusion:

> Thus it appears that the deliberate choice of an initial orientation holds an element of risk for all but very regular functions, due to the wide variations in the number of required function evaluations necessitated by even small changes in orientation. The difficulty is compounded by raising the dimensionality of the objective function since this widens the choice of orientation parameters in direct proportion.

Parkinson and Hutchinson also tested several automatic procedures which would determine appropriate initial parameters for the orientation of the simplex, but reported that none gave sufficient and regular gains over random selection to merit normal use. One suggestion they did make was that the function values at the vertices of the original simplex be used to approximate the direction of steepest descent. This information could then be used to adjust the orientation of the initial simplex so that the initial step would be along this approximation of the steepest descent direction. The difficulty of this approach, as they note, is that the quality of this estimate depends on the scale of the initial simplex. We add that this adjustment depends on the current local information about the function and thus is likely to have little impact on subsequent iterations.

Unfortunately, the same difficulties plague the multi-directional search algorithm. In fact, because the set of search directions is finite, and fixed by the choice of the initial simplex, it is all the more obvious that the orientation of the initial simplex affects the performance of the algorithm. To date, we have not spent much time addressing this problem. Since we typically use the procedure suggested by Jacoby, et al. [18] to generate the initial simplex for our test problems, we have not really experimented with changing the orientation of the simplex. One idea we would like to pursue involves restarting the procedure if it appears to be making little progress. The idea is to use the function information at each of the vertices in the simplex to construct an approximation of the Hessian at the best vertex. The approximation to the Hessian should allow us to construct a reasonable quadratic model of the problem. We could then generate a new simplex oriented along the axes of the model and restart the problem. We think this may prove to be useful when minimizing complex problems, but we have yet to implement this procedure.

There is little doubt but that the choice of the initial simplex does affect the performance of both the multi-directional search algorithm and the Nelder-Mead simplex algorithm. Any particular information about the function to be minimized may be useful in hastening the progress of the method. The only general suggestion the theory for the multi-directional search algorithm provides is that it is "safest" to start with a regular simplex. Otherwise, while the multi-directional search algorithm may be slow to converge to a solution, an unfortunate choice of scale or orientation for the initial simplex is not catastrophic. As we shall see in Chapter 5, it is not clear that the same can be said of the Nelder-Mead simplex algorithm.

4.2 Choosing the scaling factors

Underlying our discussion of the multi-directional search algorithm has been the assumption that to expand the simplex we double the lengths of the edges in the simplex while to contract the simplex we halve the lengths of all the edges. Yet, in the formal statement of the algorithm found in Section 2.3, we required only that the expansion factor μ be strictly greater than one while the contraction factor θ was required to be between 0 and 1. The convergence analysis, in particular the proof of Claim 3.3, added the restriction that μ and θ be rational numbers so that we could determine their least common denominator. Since the machine representation of any real number is, in fact, a rational number, this is indeed a very mild restriction. So why do we choose $\mu = 2$ and $\theta = \frac{1}{2}$?

Initially, these choices were based on precedent. Since the decision making process of the multi-directional search algorithm mirrors that of the Nelder-Mead simplex algorithm, it seemed reasonable to use those factors suggested by Nelder and Mead [24]. They tested a variety of choices on three small ($n = 2, 3, \text{and } 4$), but difficult, test problems. Their conclusion was that the simple strategy of either doubling or halving the step sizes was clearly the best. Parkinson and Hutchinson [26] noted that the recommendations made by Nelder and Mead were based on trials with only about one hundred combinations, so they systematically investigated several thousand combinations. They concluded that there was no general strategy which gave the best results for all the test functions. For the test functions they studied — which were again, for the most part, small, complex and limited in number — taking expansion steps of $2\frac{1}{2}$ and contraction steps of $\frac{1}{4}$ proved to be somewhat superior to the values suggested by Nelder and Mead. Walmsley [34] suggested retaining the expansion factor of 2 while setting the contraction factor to $\frac{3}{4}$ in an effort to slow the drastic contractions that often lead to difficulties with the Nelder-Mead simplex algorithm. A statistician who frequently uses the Nelder-Mead simplex algorithm in parameter estimation has remarked to me that on some of his problems an expansion factor of 3 and a contraction factor of $\frac{1}{3}$ are most effective.

As the above discussion indicates, the optimal choice of scale factors is as dependent on the function to be minimized as is the choice of the initial simplex. In fact, the two can clearly interact. This makes the choice of "optimal" scaling factors all but impossible. Again, if experience with a certain class of problems indicates that a different choice of scaling factors gives better results, it is certainly reasonable to change the scaling factors to reflect this information. In general, though, the simplicity of $\mu = 2$ and $\theta = \frac{1}{2}$ is perhaps the best argument for their choice.

One last motivation for the choice of parameters comes from a comparison, albeit strained, with the model trust region strategies for globalizing quasi-Newton methods. For an excellent discussion of globalization strategies, in general, and the model trust region strategy, in particular, see Dennis and Schnabel [14]. In some sense, the size of the current simplex, based on the step accepted in the previous iteration, gives us some measure of how much we "trust" the information we know about the function.

We could then say that the current simplex, along with its reflection, provides us with a crude "model" of the function about the current best vertex. We then use this model to determine the size of our first step. If this step produces decrease, we assume that our model may be too conservative, and thus double the size of the trust region by doubling the size of the simplex. On the other hand, if the first step does not produce decrease, we conclude that our current model is not to be trusted; we then halve the size of the trust region by halving the lengths of the edges in the simplex. This simple strategy of either doubling or halving the radius of the trust region has proven to be very effective for the model trust region methods. We hope that this same simple strategy also proves to be effective for the multi-directional search algorithm.

We close with one final comment addressed to those who are familiar with the Nelder-Mead simplex algorithm. The Nelder-Mead simplex algorithm includes a scale factor for the reflection step. In the original version of our algorithm we also included a scaling factor λ. We required the scaling factors to satisfy the following:

$$0 \; < \; \theta \; < \; 1 \; \leq \; \lambda \; < \; \mu.$$

As long as all three scaling factors are rational numbers, and $\lambda < \frac{1}{\theta}$, the convergence analysis of Chapter 3 still holds. We removed λ from our formulation of the multi-directional search algorithm based on the observation that after the first iteration, the size of the current simplex in some sense captures the function information we acquired at the previous iteration. Thus, it seems best to start the current iteration with a step of the size that was most successful in the previous iteration. Once we have accumulated additional information about the function by trying the reflection step, we then decide how to further modify the size of the steps we take. Since this implies that λ is always equal to one, there is no need to include the additional parameter λ in either the algorithm or the analysis. We note that this choice is consistent with the recommendation of Nelder and Mead and, in fact, with most implementations of the Nelder-Mead simplex algorithm.

4.3 Stopping criteria

We now turn to a discussion of how to stop the algorithm. Here, again, we have relied on the suggestions that exist for the Nelder-Mead simplex algorithm. Nelder and Mead suggested comparing the standard deviation of the function values in the simplex with a preset value and stopping when the standard deviation falls below this value. This leads to the stopping test:

$$\sqrt{\sum_{i=0}^{n} \frac{\left(f(v_i^k) - \bar{f}\right)^2}{n}} \;<\; \varepsilon\,,$$

where $\bar{f}$ is the mean of the function values at the $n+1$ vertices and ε is the preset tolerance. As Nelder and Mead observed, the success of this criterion depends on the simplex not becoming too small in relation to the curvature of the surface until the final minimum is reached.

Woods [38] showed, with two simple examples, how this stopping criterion can lead to premature termination. In the first instance, the simplex straddles a local minimizer, which means that the algorithm has successfully identified the neighborhood of a local minimizer. Unfortunately, the best vertex is at a local maximizer. His second example illustrates the reservation voiced by Nelder and Mead; if the curvature is slight the algorithm may halt prematurely. In both illustrations the difficulty lies in the fact that while the function values at the $n+1$ vertices in the simplex are "close," the simplex itself is still relatively large. Thus the algorithm halts without recognizing that the simplex has not yet identified a solution. Numerous suggestions have been made for overcoming this problem — all attempt to force the simplex to collapse before actually halting the algorithm.

Box, Davies, and Swann [4] also used the standard deviation of the function values to test for convergence, but they suggested that this test be applied differently. Instead of calculating the standard deviation before each iteration, they recommended calculating it every time some fixed number of function evaluations have been made. Two successive values of the standard deviation must be less than the specified tolerance in order to terminate the algorithm. In addition, the corresponding mean values of the function (at the $n+1$ vertices) are not allowed to differ by more than some specified amount. Their goal is to ensure that the search continues until the simplex has collapsed — ideally onto a minimizer.

Parkinson and Hutchinson [26] suggested that the stopping criteria should restrict both the range in f and the corrections to v_i for all i. They proposed the following two tests:

$$f(v_n^k) - f(v_0^k) \quad < \quad \varepsilon \, ,$$

where v_n^k is the "worst" vertex in the simplex, i.e., the vertex with the largest function value, and

$$\frac{1}{n} \sum_{i=0}^{n} \left\| v_i^{k+1} - v_i^k \right\|^2 \quad < \quad \epsilon \, . \tag{4.2}$$

Woods [38] interpreted the second stopping criterion proposed by Parkinson and Hutchinson (4.2) as a measure of how far the simplex has moved. He then noted that the distance the simplex moves is related to the size of the simplex, and on the basis of this observation proposed that the size of the simplex be used as a stopping criterion. This leads to the following test:

$$\frac{1}{\Delta} \max_{1 \leq i \leq n} \left\| v_i^k - v_0^k \right\| \quad \leq \quad \epsilon \, , \tag{4.3}$$

where $\Delta = \max\left(1, \|v_0^k\|\right)$. This would then measure the relative size of the simplex by considering the length of the longest edge adjacent to v_0^k.

We found the test proposed by Woods the most appealing since it promised to circumvent both of the difficulties he illustrated. Furthermore, in the multi-directional search algorithm, the length of each edge adjacent to the best vertex defines a relative step of length "one" in the direction determined by that edge. Thus, using (4.3) to test for convergence in the multi-directional search algorithm is equivalent to the "step tolerance" test proposed by Dennis and Schnabel [14] for the quasi-Newton methods. This test measures whether the algorithm has ground to a halt, either because it has stalled or converged, by imposing a measure of the relative change in the successive values of v_0^k. Dennis and Schnabel also suggested a reasonable guideline for choosing ϵ: if p significant digits of x_* are desired, ϵ should be set to 10^{-p}.

Our current implementations of both the multi-directional search algorithm and the Nelder-Mead simplex algorithm, which were used to generate the performance results discussed in Chapter 5, use (4.3) as a stopping criterion. In addition, we limit the total number of iterations — though, as will be seen, our tolerance for this criterion is currently very generous. In many settings it may be more appropriate to limit the total number of function evaluations rather than the total number of iterations.

Our overall experience using the "step tolerance" as a stopping criterion has been favorable. In reviewing our test results, however, we noted that once the multi-directional search algorithm identifies a solution, it often seems to spend a great many iterations contracting before it finally satisfies (4.3). This should not be too surprising. The multi-directional search algorithm is really a gradient related method. As a consequence, it has good global properties, but it is slow to converge locally. The question, then, is whether or not the user is interested in identifying the solution accurately, or in identifying the neighborhood of a solution. As Parkinson and Hutchinson [26] noted, the stopping criteria "should normally be selected according to the objective required, e.g. position or value of the minimum."

We have now decided that as in the implementations of quasi-Newton methods, it is probably best to test both some measure of the length of the steps taken as well as some measure of the function values. With this in mind, we are interested in testing a "function tolerance" criterion of the form:

$$\frac{1}{\delta} \max_{1 \leq i \leq n} \frac{\|f(v_i^k) - f(v_{r_i}^k)\|}{2\,\|v_i^k - v_{r_i}^k\|} \;\leq\; \varepsilon\,, \tag{4.4}$$

where $\delta = \max\left(1, |f(v_0^k)|\right)$. We must confess that we have not yet implemented this idea, so (4.4) may need further modification. The appeal of the function tolerance test, however, lies in the fact that it attempts to make use of the function information we compute at each iteration. Furthermore, since the best vertex v_0^k is "centered" between the original simplex and the reflected simplex, this should give us some information about the neighborhood in which v_0^k lies.

4.4 Our choices

As we have seen in the preceding discussion, there are no easy answers when it comes to choosing the initial simplex, the scaling factors, or the stopping criteria. In summary, we made the following choices to run the tests we report in the next chapter:

To generate an initial simplex, from a given starting point, we use the simple procedure suggested in Jacoby, Kowalik and Pizzo [18] to generate a regular simplex. Since the problems we tested were not badly scaled, we chose to start with edges of length one. The orientation of the simplex is fixed by the procedure given in [18].

Our choice of scaling factors is consistent with our discussion of the algorithm in Chapter 2; we set $\mu = 2$ and $\theta = \frac{1}{2}$.

Finally, our current implementation uses the test suggested by Woods to stop the algorithm. This choice, (4.3), measures the relative size of the simplex by considering the length of the longest edge adjacent to v_0^k and stops when this measure falls below the given tolerance. It should be noted, however, that we think an additional test, based on function information, needs to be added to prevent additional iterations when it should be clear that a solution has been identified.

Chapter 5

Performance

From the beginning of this research, we have been interested in developing an algorithm that is practical. First and foremost the method needed to be robust; we wanted an algorithm for which there was reasonable certainty that the answer returned was, in fact, a local minimizer of the function. Our convergence analysis led us to believe that such a claim was possible for the multi-directional search algorithm. Our performance results would seem to confirm this conjecture. We also wanted an algorithm that would work well when the function values were “noisy,” by which we mean that the function values are subject to error. Again, our convergence analysis suggested that noise in the function values would not have a detrimental effect on the progress of the search — another conjecture which our performance results would seem to sustain.

Before presenting our test results, we would like to discuss both the problems we tested and the factors we varied.

5.1 Preliminaries

We believe that ultimately the multi-directional search algorithm will be most interesting when implemented on parallel machines. As we have already noted in Chapter 1, the two main costs in numerical optimization procedures are the solution of linear systems of equations and the evaluation of functions. These are the two areas where parallel computation appears to be most beneficial in numerical optimization. There are no linear systems of equations to be solved for the multi-directional search algorithm. On the other hand, the n function evaluations required at each step of the multi-directional search algorithm can be computed concurrently, making the algorithm ideally suited to parallel computer architectures.

To test our algorithm, we used standard problems from the optimization literature. None of these problems is expensive to compute, and all of them possess easily computed analytic derivatives. Moreover, many of these problems are relatively small — of dimension two, three, and four. Nevertheless, they will give us some indication of the performance of the algorithm, if not of its competitiveness.

One way to simulate expensive function evaluations is to write function evaluation routines which perform large numbers of superfluous floating point operations. We have used this trick in the past to generate "representative" timing results for a particular parallel machine. For our purposes, however, these tests were not very interesting since they really provided more information about the machine than about the algorithm. We have little interest in establishing benchmarks of the performance characteristics of a given machine. Thus, no timing results will be reported.

Given that we are not interested in timing results, per se, we decided to use a different measure of performance. Since our goal is to reduce the time spent computing function values, we counted the number of *actual* function evaluations required to return a solution for the specified tolerance level and then concocted a measure of the *effective* number of function evaluations. To derive this measure we simply noted that if we had twenty processors, and the dimension of the problem was thirty-two, then we could "effectively" compute all the function evaluations in the time it would take to compute two function values on a sequential machine. This gives an effective measure of two. Our current implementation of the multi-directional search algorithm makes no attempt to address the issue of load balancing, though we will have more to say on this subject later. The machine we used to run our tests was a Sequent Symmetry with twenty processors, but we could just as easily have run our tests on a sequential machine. In fact, our results indicate that there may be some argument for using this algorithm on sequential machines as well.

5.1.1 The competition

For the sake of comparison we used three other algorithms to test the same set of problems:

The Nelder-Mead simplex algorithm

We chose the Nelder-Mead simplex algorithm because in some sense this is the most reasonable direct search method against which to compare our algorithm. Both algorithms are direct search methods which use a simplex to determine the search direction and the size of the step. Both algorithms use the same decision-making process in accepting or rejecting trial steps. Neither algorithm requires derivatives, and, finally, both algorithms could be implemented with identical stopping criteria. We were also interested in the Nelder-Mead simplex algorithm because it is so widely used. Margaret Wright has stated that over fifty percent of the calls received by the support group for the NAG software library concerned the version of the Nelder-Mead simplex algorithm to be found in that library.

We programmed our own version of the Nelder-Mead simplex algorithm. The only modification we made to the algorithm as originally specified by Nelder and Mead was in the acceptance test for the expansion step. The original algorithm required that the function value at the expanded vertex be better than the function value at the best vertex, i.e., $f(v_e^k) < f(v_0^k)$. We instituted a stricter test; we only accepted the expanded vertex if the function value at that vertex was better than the function value at the reflected vertex, i.e., $f(v_e^k) < f(v_r^k) < f(v_0^k)$. This modification was made in an effort to slow distortions to the shape of the simplex, for reasons that we shall discuss later.

The method of steepest descent

Our convergence analysis made clear the relation between the multi-directional search algorithm and gradient related methods. The multi-directional search algorithm could, in fact, be viewed as a crude approximation of the method of steepest descent with finite-difference approximations to the gradient. As a consequence, we thought that a comparison between the two methods would be interesting. Thus, while the problems we tested did have analytic derivatives, we used finite difference approximations to the gradient to determine the search direction. We used line searches to globalize the method. Rather than write our own code, we used the UNCMIN code, which is described in Appendix A of Dennis and Schnabel [14] and is available from Robert B. Schnabel at the University of Colorado.

A quasi-Newton method

Finally, we compared our results to those found using a quasi-Newton method, since these are the methods of choice in the numerical optimization community. As with steepest descent, we did not use analytic derivatives; the gradient was derived using finite difference approximations and the Hessian was updated using BFGS. We also used a line search strategy to globalize the method since this strategy seemed most consistent with the other three algorithms. Again, we used the UNCMIN code.

5.1.2 The test problems

Our original tests included such classic problems as the Beale function, the Helical Valley function, and the Wood function, which are of dimension two, three, and four, respectively. However, the more interesting results came from the problems for which we could vary the dimension since the behavior of the two simplex methods as n was increased proved to be most illuminating. As a consequence, we will devote our discussion to test problems with variable dimension from the Moré, Garbow, and Hillstrom problem set [22]. These problems are given below. Note that for all six problems the function to be minimized is defined to be

$$f(x) \;=\; \sum_{i=1}^{m} f_i(x)^2 \,.$$

1. *Penalty function I*

(a) n variable, $\quad m = n + 1$

(b) $f_i(x) = 10^{-\frac{5}{2}}(x_i - 1), \qquad 1 \le i \le n$

$f_{n+1}(x) = \left(\sum_{j=1}^{n} x_j^2\right) - \frac{1}{4}$

(c) $x_0 = (\xi_j) \quad$ where $\quad \xi_j = j$

(d) $f_* = 2.24997 \cdots 10^{-5} \qquad$ if $n = 4$

$f_* = 7.08765 \cdots 10^{-5} \qquad$ if $n = 10$

2. *Extended Powell singular function*

(a) n variable but a multiple of 4, $\quad m = n$

(b) $f_{4i-3}(x) = x_{4i-3} + 10x_{4i-2}$

$f_{4i-2}(x) = 5^{\frac{1}{2}}(x_{4i-1} - x_{4i})$

$f_{4i-1}(x) = (x_{4i-2} - 2x_{4i-1})^2$

$f_{4i}(x) = 10^{\frac{1}{2}}(x_{4i-3} - x_{4i})^2$

(c) $x_0 = (\xi_j) \quad$ where $\quad \xi_{4j-3} = 3, \;\; \xi_{4j-2} = -1, \;\; \xi_{4j-1} = 0, \;\; \xi_{4j} = 1$

(d) $f_* = 0 \quad$ at $\quad$ the origin

3. *Extended Rosenbrock function*

(a) n variable but even, $\quad m = n$

(b) $f_{2i-1}(x) = 10(x_{2i} - x_{2i-1}^2)$

$f_{2i}(x) = 1 - x_{2i-1}$

(c) $x_0 = (\xi_j) \quad \text{where} \quad \xi_{2j-1} = -1.2, \quad \xi_{2j} = 1$

(d) $f_* = 0 \quad \text{at} \quad (1, \ldots, 1)$

4. *Trigonometric function*

(a) n variable, $\quad m = n$

(b) $f_i(x) = n - \sum_{j=1}^{n} (\cos x_j + i(1 - \cos x_i) - \sin x_i)$

(c) $x_0 = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$

(d) $f_* = 0$

5. *Variably dimensioned function*

(a) n variable, $\quad m = n + 2$

(b) $f_i(x) = x_i - 1, \qquad i = 1, \ldots, n$

$f_{n+1}(x) = \sum_{j=1}^{n} j(x_j - 1)$

$f_{n+2}(x) = \left(\sum_{j=1}^{n} j(x_j - 1)\right)^2$

(c) $x_0 = (\xi_j) \quad \text{where} \quad \xi_j = 1 - \frac{j}{n}$

(d) $f_* = 0 \quad \text{at} \quad (1, \ldots, 1)$

6. *The perfect function (the l^2-norm)*

(a) n variable, $\quad m = n$

(b) $f_i = x_i$

(c) $x_0 = (10, \ldots, 10)$

(d) $f_* = 0 \quad \text{at} \quad \text{the origin}$

Note that the last problem is not in the Moré, Garbow, and Hillstrom problem set, but we included it as a "benchmark" problem; its inclusion produced a most unexpected result.

5.1.3 The questions to be answered

Our testing was designed to address the following questions:

- What happens as we vary the dimension of the problem?
- What happens as we allow increasingly smaller step sizes?

At this point we assume that the function evaluations are "noise-free," i.e., the function values are accurate to machine precision. The quasi-Newton method was clearly the winner in this case, which should be no surprise. The interesting comparisons here are between the other three methods, as we shall see in Section 5.2.

To even up the race, we then asked the following question:

- What happens if we introduce random error into the function values?

Varying the dimension

Varying the dimension of the test problems was straightforward. Since the extended Rosenbrock function requires that n be even, and the extended Powell singular function requires that n be a power of four, we limited our tests accordingly. We started all problems with $n = 2$ (except for the extended Powell singular function) and ended all problems with $n = 40$. Intermediate values for n typically included 4, 8, 16, 20, and 32.

Decreasing the step size

Allowing increasingly smaller step sizes was also straightforward, but requires a little more explanation. In Section 4.3 we discussed, at some length, the choice of stopping criteria for the multi-directional search algorithm. Our final suggestion was a test that measured the relative size of the simplex by considering the length of the longest edge adjacent to the best vertex:

$$\frac{1}{\Delta} \max_{1 \leq i \leq n} \left\| v_i^k - v_0^k \right\| \;\leq\; \epsilon\,, \tag{5.1}$$

where $\Delta = \max\left(1, \|v_0^k\|\right)$. As we noted, since the length of each edge adjacent to the best vertex defines a relative step of length "one," this test could then be viewed as a step tolerance test. Thus, ever decreasing values of ϵ allow for ever decreasing step sizes.

We allowed ϵ to vary from 0.10D-01 to 0.10D-07. We began our tests with $\epsilon =$ 0.10D-01 and proceeded until (5.1) was satisfied. We then divided ϵ by 10 and restarted the algorithm with our current solution. This same procedure was used for the Nelder-Mead simplex algorithm, though it should be noted that the connection between the stopping criterion (5.1) and the size of the step is not quite as clear since the size of the step depends on the shape of the simplex as well as on the lengths of the edges.

The UNCMIN code we used for the steepest descent method and the quasi-Newton method employs both a "step tolerance" and a "gradient tolerance" convergence test. For a further discussion of these two tests, see Dennis and Schnabel [14]. Since we were most interested in stopping when the step sizes became small enough, we varied the step tolerance constant ϵ as described above. To minimize the effect of the gradient tolerance test, we set its tolerance constant to machine precision. It should be noted, however, that in many of our experiments, as ϵ became smaller and smaller, the UNCMIN routines stopped because the gradient tolerance test had already been satisfied. This biases our results in favor of the methods tested using the UNCMIN routines.

One final point to be made is that the tests using the UNCMIN routines were restarted from the initial point x_0 for each new value of ϵ. We did not simply restart the procedure with the current solution, as we did for the simplex methods. This allowed us to get fair counts — without duplication — for the number of function evaluations required to converge to a solution for the given value of ϵ. This difference explains the seeming anomalies in the runs for which random errors were introduced in the function values.

Finally, all four methods were tested in double precision. This allowed us a reasonable range over which to vary our step tolerance parameter ϵ.

Introducing random errors

One of our aims was to develop an algorithm that worked well in the presence of noise. No standard suite of test problems exists for noisy functions. Consequently, we elected to repeat our first round of experiments and simply add random perturbations to the function values.

To simulate "noisy" function values, we first calculated the value of f at the point x. Then, we reassigned the value of f as follows:

$$f(x) \quad \leftarrow \quad f(x) + \max\left\{\rho \cdot |f(x)|\,,\, \eta\right\} \cdot \mu\,,$$

where μ is a random number with uniform distribution on $[-1, 1]$ and ρ and η are parameters set by the user. To obtain μ we used the random number generator on the Sequent Symmetry. For the test results we will show, ρ and η were both equal to 0.10D-03.

One final point to make is that for both the steepest descent method and the quasi-Newton method, the finite difference approximations to the gradient were computed sensibly. The UNCMIN code specifically asks for the number of good digits in the optimization function routine. Our answer to this question was commensurate with the degree of error we were introducing to the function values. Thus, the tests were not biased against the finite difference approximations to the gradient.

Other than the modification to the value returned by the function evaluation routines, and the corresponding change in the number of good digits, the tests with random noise were run exactly as described above.

5.2 Results

The most interesting result we have to report is, in some ways, the most unexpected: The Nelder-Mead simplex algorithm is simply not robust. In the course of our research, the convergence analysis for the multi-directional search algorithm revealed that the uniform linear independence of the search directions assured us that at least one search direction was uniformly bounded away from being orthogonal to the gradient. On the other hand, we could never prove that the search directions chosen by the Nelder-Mead simplex algorithm were uniformly bounded away from being orthogonal to the gradient. Thus we asked what happens to the angle between the gradient at the worst vertex v_n, the vertex from which the Nelder-Mead simplex algorithm searches, and the search direction generated by the algorithm. The answer was startling.

On *every* test problem for which the dimension of the function could be increased, the very pathology we cannot prevent in the Nelder-Mead simplex algorithm actually occurs: The search direction becomes orthogonal to the gradient at the point from which we search. The disconcerting consequence is that the answer v_0^* returned by the algorithm is not a solution. This phenomenon is demonstrated in Tables 5.1–5.6.

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.70355D-04	1605	89.396677792198
.10D-02	.62912D-04	3360	89.935373548613
.10D-03	.62912D-04	3600	89.994626919197
.10D-04	.62912D-04	3670	89.999288284747
.10D-05	.62912D-04	3750	89.999931862232
.10D-06	.62912D-04	3872	89.999995767877
.10D-07	.62912D-04	3919	89.999999335010

Table 5.1 Nelder-Mead on Penalty function I with $n = 8$

Note that this deterioration of the search direction occurs even for the "perfect" function $x^T x$, as can be seen in Table 5.6. In fact, the Nelder-Mead simplex algorithm was halted before completing the table because the maximum number of iterations allowed (300,000) had been reached. Also observe that the negative gradient at v_n^* and the search direction become *increasingly* orthogonal with each decrease in the size of the step tolerance ϵ. Moreover, this unfortunate behavior appears at *different* dimensions for different problems — as early as $n = 8$ for the Penalty function I (Table 5.1) and as late as $n = 40$ for the Trigonometric function (Table 5.4). Finally, note that this deterioration continues for every choice of n greater than that at which it first occurs, as we see for the extended Rosenbrock function in Tables 5.3 and 5.7, and for $x^T x$ in Tables 5.6 and 5.8. These observations raise serious doubts as to the

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.49925D-02	24791	89.385723944024
.10D-02	.23830D-02	43986	89.933941414922
.10D-03	.23749D-02	66848	89.991402723733
.10D-04	.15441D-02	125781	89.999000208078
.10D-05	.15441D-02	127998	89.999911526521
.10D-06	.15441D-02	140394	89.999989953855
.10D-07	.15441D-02	163009	89.999999795613

Table 5.2 Nelder-Mead on extended Powell singular function with $n = 32$

applicability of the Nelder-Mead simplex algorithm for all but the smallest values of n.

Popular wisdom has long stated that the Nelder-Mead simplex algorithm is inefficient for problems with a "large" number of variables, say $n \geq 10$, but we were not aware of any satisfactory explanations as to why this was so. Our numerical results would indicate that not only is the Nelder-Mead simplex algorithm inefficient on large problems, it is also unreliable. Of most concern should be the fact that we cannot predict at what dimension the search directions will deteriorate.

While we have demonstrated *how* the Nelder-Mead simplex algorithm deteriorates in higher dimensions, we still have not explained *why* this is so. We do not yet have an answer to this question, but we do have a conjecture as to the probable cause. We

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.34838D+02	193	69.428066137874
.10D-02	.83903D+01	4868	89.941249866308
.10D-03	.11080D+01	19764	89.966823864945
.10D-04	.11080D+01	20084	89.997448235539
.10D-05	.11080D+01	20232	89.999747777662
.10D-06	.11080D+01	20443	89.999968762310
.10D-07	.11080D+01	20574	89.999996161174

Table 5.3 Nelder-Mead on extended Rosenbrock function with $n = 16$

believe the problem can be traced back to the interaction between the size and the shape of the simplex used by the Nelder-Mead simplex algorithm.

As we noted in Chapter 4, the multi-directional search algorithm automatically rescales the entire simplex if it is either too large or too small. The Nelder-Mead simplex algorithm, however, only rescales the entire simplex as a last resort. If no improvement can be found by taking any other step, the algorithm takes a "shrink" step — which is equivalent to the contraction step of the multi-dimensional search algorithm. It is an easy exercise to show that if the original simplex and its reflection (as specified by the Nelder-Mead simplex algorithm) are in a region where the function is convex, then the Nelder-Mead simplex algorithm will *not* consider the shrink step. The proof is similar to that given for Lemma 3.1. In fact, when we ran the Nelder-Mead simplex algorithm on all the problems listed in Section 5.1.2, we found that even when the functions were *not* convex, only 33 out of some 2.9 *million* iterations resulted

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.12469D+00	4565	78.427653672869
.10D-02	.21610D-02	13150	80.680288863396
.10D-03	.62321D-04	33436	84.899823503396
.10D-04	.19584D-04	62190	89.129939277735
.10D-05	.65252D-05	176526	89.852022558573
.10D-06	.64708D-05	213009	89.984435547320
.10D-07	.64699D-05	259611	89.998494778414

Table 5.4 Nelder-Mead on Trigonometric function with $n = 40$

in a shrink step. Thus, if the initial simplex is too large, the Nelder-Mead simplex algorithm only contracts one vertex at each iteration — and each contraction results in a distortion of the initial simplex. As the dimension of the problem grows, resizing the entire simplex requires more and more iterations, each of which is conducted independently. As a result, the distortion of the simplex increases with the dimension of the problem. This distortion, in turn, means that it is ever more likely that the search directions will deteriorate.

This conjecture is consistent with observations made by both Parkinson and Hutchinson [26] and Walmsley [34]. Parkinson and Hutchinson, in fact, suggest

> ... reducing the incidence of numerous successive contractions. In our experience the latter phenomenon occurs with NMS [the Nelder-Mead simplex algorithm] when a drastic rescaling of the simplex is needed in order to change substantially the direction of search, and can even give rise to apparent convergence at a false minimum.

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.24123D+00	338	84.219810666003
.10D-02	.44628D-01	1680	88.913585473601
.10D-03	.26354D-01	5249	89.880567806884
.10D-04	.13416D-01	8166	89.982604055225
.10D-05	.12678D-01	15901	89.998395072707
.10D-06	.12678D-01	16190	89.999852203339
.10D-07	.12678D-01	16781	89.999981760597

Table 5.5 Nelder-Mead on Variably dimensioned function with $n = 16$

By contrast, the multi-directional search algorithm, while slow to converge, is very reliable. Consider, for example, its behavior on the extended Rosenbrock function at the same dimension for which the search deteriorates in the Nelder-Mead simplex algorithm. The results are contained in Tables 5.3 and 5.9. The extended Rosenbrock function, which is a very hard problem to solve, does prove to be difficult for the multi-directional search algorithm. However, while the multi-directional search algorithm does require both small steps and a great many function evaluations to reach a solution, the final answer returned by the algorithm (for $\epsilon = 0.10\text{D-}07$) is correct to the square root of machine precision, which is certainly a respectable showing. Furthermore, this behavior is consistent across all the choices of n for which we tested the method on this problem, as shown in Table 5.10.

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.27091D+00	38601	89.509783891566
.10D-02	.23877D+00	65475	89.954928916223
.10D-03	.23877D+00	71874	89.995057909425
.10D-04	.23461D+00	175010	89.999471407025
.10D-05	.23461D+00	195185	89.999954253610
......	.23461D+00	353670	89.999410272082

Table 5.6 Nelder-Mead on $x^T x$ with $n = 32$

The reliability of the multi-dimensional search algorithm also holds for the "perfect" function. Again, we look at the dimension for which the search direction deteriorates in the Nelder-Mead simplex algorithm, as shown in Tables 5.6 and 5.11. The final answer returned by the algorithm (for $\epsilon = 0.10\text{D-}07$) is correct to machine precision. Table 5.12 shows that this behavior is again consistent across all the choices of n for which we tested the method on this problem.

For the method of steepest descent and the quasi-Newton function, the "perfect" function $x^T x$ is trivial to solve; with exact representations for the gradient, both methods solve the problem in one step. With finite difference approximations to the gradient, both methods require two iterations to solve the problem. We include Tables 5.13 and 5.14 to show how both of these methods perform on the more difficult extended Rosenbrock function. The difficulty of the extended Rosenbrock function can be seen in the fact that the steepest descent method required over 100,000 function evaluations to converge to a solution, although this is still much better than the num-

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.43187D+02	236	37.350441344532
.10D-02	.41925D+01	9726	89.711342380932
.10D-03	.38871D+01	13141	89.993225430277
.10D-04	.36634D+01	18743	89.993332178692
.10D-05	.36634D+01	19429	89.999914206721
.10D-06	.36634D+01	19824	89.999994959976
.10D-07	.20803D+01	59303	89.999999706363

Table 5.7 Nelder-Mead on extended Rosenbrock function with $n = 20$

ber required for the multi-directional search algorithm. The quasi-Newton method demonstrated very clearly its claim as the thoroughbred of optimization methods.

The interesting question then becomes, what happens when we introduce random noise into the function values to even up the race. As we have already noted, for these tests we introduced error on the order of 0.10D-03, which means that we should expect three significant digits in the solutions returned by each of the algorithms. We will limit our discussion to the function $x^T x$ and show what happens for each of the four methods when $n = 16$.

The Nelder-Mead simplex algorithm returned the same answer, to five significant digits, regardless of the step size. Unfortunately, the answer returned was only correct in the first two digits, rather than in the first three digits we expected. Note also that the angle between the negative gradient at v_n^* and the search direction stays dangerously close to 90°. Even more disturbing is the fact that this behavior is now

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.17176D-01	48459	89.287624428318
.10D-02	.13606D-01	118057	89.893411000844
.10D-03	.13598D-01	141548	89.991891633507
.10D-04	.13598D-01	166018	89.999136336676
.10D-05	.13598D-01	200592	89.999918332639
.10D-06	.13598D-01	327426	89.999988870646
......	.13598D-01	345475	89.999992334691

Table 5.8 Nelder-Mead on $x^T x$ with $n = 40$

occurring for $n = 16$, rather than $n = 32$ when we do not add random perturbations. Thus the problem we have seen with Nelder-Mead only becomes worse when we add "noise" as a complicating factor.

Again, the multi-directional search algorithm proved to be robust. In Table 5.16 we can see that we have at least three significant digits in the solution, regardless of the step size. More promising is the fact that we no longer see significant jumps in the number of function calls required to converge to a solution as we decrease the step size.

Finally, we look at the performances of the steepest descent method and the quasi-Newton method once we have introduced random perturbations. As can be seen in Table 5.17, the steepest descent method returned only two significant digits when

step tolerance	$f(v_0^*)$	function calls actual	function calls effective
.10D-01	.36169D+02	160	10
.10D-02	.36131D+02	320	20
.10D-03	.92623D-01	53680	3355
.10D-04	.22256D-02	151888	9493
.10D-05	.37254D-04	331232	20702
.10D-06	.72650D-06	579808	36238
.10D-07	.30764D-08	904880	56555

Table 5.9 Multi-directional search on extended Rosenbrock function with $n = 16$

$\epsilon = 0.10$D-03. In Table 5.18 we see that the quasi-Newton method failed to return the correct answer in two cases.

5.3 Conclusions

Our preliminary tests of the multi-directional search algorithm have led us to the following conclusions:

First, the Nelder-Mead simplex algorithm is *not* robust. The convergence proof for the multi-directional search algorithm led to a revelation of how Nelder-Mead fails on large problems. What we find troubling is that "large," in this context, cannot be predicted a priori. For the Penalty function I, for instance, "large" means $n \geq 8$. Furthermore, when we add random perturbations to the function values, the Nelder-Mead simplex algorithm deteriorates even sooner. There is no question but that the

n	$f(v_0^*)$	function calls actual	function calls effective
2	.22725D-09	20870	10435
4	.35718D-09	67748	16937
8	.18606D-08	235032	29379
16	.30764D-08	904880	56555
20	.17643D-07	1335220	66761
32	.22632D-07	3127520	195470
40	.27874D-07	4975480	248774

Table 5.10 Multi-directional search on extended Rosenbrock function with step tolerance ϵ = 0.10D-07

Nelder-Mead simplex algorithm is usually faster than the multi-dimensional search algorithm when the problems are small, i.e., of dimension 2, 3, or 4, but we would caution anyone using this algorithm, particularly in an experimental setting, to be wary of the answers returned by the algorithm when the problems are any larger.

We have also seen that the multi-directional search algorithm is slow but steady, even on moderately large problems. However, we must confess that the almost one million *actual* function evaluations required to solve the extended Rosenbrock function for $n = 16$ and ϵ = 0.10D-07 (see Table 5.9) is unacceptably high. In the case of the extended Rosenbrock function, we know of at least one reason why the algorithm requires so many function evaluations. As we discussed in Chapter 4, if the simplex used to start the procedure is too large, the multi-directional search algorithm will automatically rescale the simplex. However, this procedure may prove to be quite

step tolerance	$f(v_0^*)$	function calls actual	function calls effective
.10D-01	.46878D-04	14176	886
.10D-02	.54975D-06	17536	1096
.10D-03	.11928D-07	21312	1332
.10D-04	.13482D-10	25984	1624
.10D-05	.28071D-11	27136	1696
.10D-06	.11419D-13	32832	2052
.10D-07	.49835D-16	37632	2352

Table 5.11 Multi-directional search on $x^T x$ with $n = 32$

costly. We have carefully traced the steps taken by the multi-directional search algorithm for the extended Rosenbrock function when $n = 2$. What we have discovered is that the algorithm takes hundreds of contraction steps before the simplex is small enough to allow the search to make any progress. In Section 4.1.2 we discussed a simple way to accelerate these contractions. The overhead required for this acceleration is nominal and the net effect could lead to significant savings in the number of function evaluations required by the algorithm.

Another observation, also made in Chapter 4, is that the multi-directional search algorithm uses a step tolerance test to determine whether or not to stop the search. While this test certainly works, we believe that we should also implement another test that makes use of the function information we acquire at each iteration to decide

n	$f(v_0^*)$	function calls actual	function calls effective
2	.23534D-17	236	118
4	.21075D-16	716	179
8	.97194D-16	2704	338
16	.12164D-16	8848	553
20	.18796D-16	13580	679
32	.49835D-16	37632	2352
40	.21544D-15	58160	2908

Table 5.12 Multi-directional search on $x^T x$ with step tolerance ϵ = 0.10D-07

whether or not the algorithm is near a solution. One such "function" tolerance test was discussed in Section 4.3.

Finally, we note that our current implementation of the multi-directional search algorithm does not yet exploit the full capabilities of a multi-processor machine. We have several interesting idea for making better use of multiple processors but we will defer such discussions to the next chapter.

In closing, let us say that we believe we have the start of a promising algorithm. Our preliminary performance results give us reason to believe that the multi-directional search algorithm may prove to be most useful when the function evaluations are subject to error. We intend to devote some time to modifying the algorithm to make it more efficient. We still have many ideas to explore.

step tolerance	$f(v_0^*)$	function calls
.10D-01	.16538D+01	80
.10D-02	.16459D+01	160
.10D-03	.22155D-01	98800
.10D-04	.27790D-04	159857
.10D-05	.14400D-07	161474
.10D-06	.47005D-11	161778
.10D-07	.45073D-13	161912

Table 5.13 Steepest descent on extended Rosenbrock function with $n = 16$

step tolerance	$f(v_0^*)$	function calls
.10D-01	.16114D+01	90
.10D-02	.25461D-03	503
.10D-03	.13001D-06	592
.10D-04	.85994D-09	762
.10D-05	.13750D-09	861
.10D-06	.13596D-09	896
.10D-07	.40865D-14	1228

Table 5.14 Quasi-Newton method on extended Rosenbrock function with $n = 16$

step tolerance	$f(v_0^*)$	function calls	the angle between $-\nabla f(v_n^*)$ and the search direction
.10D-01	.55419D-02	2999	89.531667319177
.10D-02	.55354D-02	3077	89.650748831041
.10D-03	.55354D-02	3159	94.399435839573
.10D-04	.55354D-02	3249	93.492650826296
.10D-05	.55346D-02	3370	91.759887037185
.10D-06	.55346D-02	3449	91.050583068438
.10D-07	.55345D-02	3550	88.185598124193

Table 5.15 Nelder-Mead with noise on $x^T x$ with $n = 16$

step		function calls	
tolerance	$f(v_0^*)$	actual	effective
.10D-01	.12469D-03	4176	261
.10D-02	.88067D-04	4384	274
.10D-03	.88067D-04	4480	280
.10D-04	.88067D-04	4608	288
.10D-05	.88067D-04	4704	294
.10D-06	.87835D-04	4816	301
.10D-07	.87835D-04	4944	309

Table 5.16 Multi-directional search with noise on $x^T x$ with $n = 16$

step tolerance	$f(v_0^*)$	function calls
.10D-01	-.56422D-04	204
.10D-02	-.85780D-04	222
.10D-03	.38304D-02	72
.10D-04	-.87355D-04	245
.10D-05	-.49252D-04	250
.10D-06	-.71561D-04	216
.10D-07	-.94555D-04	183

Table 5.17 Steepest descent with noise on $x^T x$ with $n = 16$

step tolerance	$f(v_0^*)$	function calls
.10D-01	.38179D-02	69
.10D-02	-.14000D-04	186
.10D-03	-.74024D-04	205
.10D-04	-.50811D-04	315
.10D-05	.13134D-02	265
.10D-06	-.85300D-04	263
.10D-07	.48307D-03	387

Table 5.18 Quasi-Newton method with noise on $x^T x$ with $n = 16$

Chapter 6

Future research

Before we discuss where we would like to go next, let us recall where we have been. We have accomplished the following:

- We have developed a new, simple, robust direct search algorithm.
- We have derived a convergence theorem for the new algorithm.
- We have determined how the Nelder-Mead simplex algorithm fails on large problems.

But this list also points to work that remains to be done. As we noted in Chapter 5, while the multi-directional search algorithm has several attractive features, it requires further refinement before it is efficient enough to be considered a competitive algorithm. There are numerous directions to explore. First, we have only just begin to explore the potential of the algorithm when implemented on parallel machines. Second, we have a wealth of ideas to pursue in an effort to both speed up the progress of the algorithm and to extend its applicability. Many of these improvements were discussed in Chapters 4 and 5. We are also interested in tackling more difficult problems; for instance, problems where the function is not differentiable. Thus, we have several ideas for dealing with such cases.

The convergence theorem for the multi-directional search algorithm has suggested several interesting research directions. First, we believe the convergence proof can be modified to handle the case where the function is not differentiable. Second, we believe that the convergence result can be developed into a convergence theory for an entire class of direct search algorithms. We believe that the same observations that helped us identify *how* the Nelder-Mead simplex algorithm fails on large problems will also help us determine *why* the Nelder-Mead simplex algorithm fails. This is of interest to us since the adaptive properties of the Nelder-Mead simplex algorithm clearly give it a competitive edge over the multi-directional search algorithm when the dimension of the problem is no more than four. If we understood why the search breaks down in higher dimensions, and could insure that this breakdown did not occur, we might be able to adopt some of these properties to speed the progress of the multi-dimensional search algorithm. Finally, the convergence theorem demonstrated that the multi-directional search algorithm enforces step length control without resorting to the Armijo-Goldstein-Wolfe conditions. We would like to understand both why this is so and how it relates to the existing theory for other unconstrained optimization methods.

We will now discuss several of these topics in further detail

6.1 Refining the algorithm

6.1.1 Parallelism

As it currently stands, the parallelism in the multi-directional search algorithm is straightforward. The algorithm reduces to four `DO` loops that can be executed in parallel: the loop to initialize the procedure and the three loops for each of the reflection, expansion, and contraction steps. (See the formal statement of the algorithm given in Section 2.3.) We have not yet addressed the issue of load balancing. So far we have ignored processors that may be idle during the course of the computation, as our results in Chapter 5 should make clear. The proof of our convergence theorem, however, has given us several ideas for improving the performance of the multi-directional search algorithm on parallel machines.

Returning to the convergence theorem, note that we only require *simple* decrease; the function value of *one* of the new vertices must be better than the function value at the best vertex. We systematically search in n linearly independent directions because if f is differentiable, one of these search directions is guaranteed to produce a direction of descent. However, as soon as the algorithm identifies a vertex that has a function value that is better than the function value at the best vertex, the search can be stopped. This observation has led us to the following load balancing strategies, given p available processors:

First, assume that $n > p$. Begin the iteration by searching in p directions. If one of the vertices returns a function value that is better than the function value at the best vertex, then the algorithm can stop the search, update all relevant information and go on to the next iteration — there is no need to search in all n directions. If none of the p trial vertices can replace the current best vertex, choose p more search directions from the set of n search directions. The search continues in this way until either the best vertex is replaced, or all n search directions have been considered. Clearly some care must be taken in choosing the order in which the n possible search directions are considered. Ideally, we would like to limit the search to the first p directions considered at each iteration, but we must also be careful to cycle through the directions so that the algorithm does not simply minimize over a subspace. One important point to make is that if we can find reasonable strategies for determining the order of the search directions, we may have an algorithm that is also useful on *sequential* machines — p simply equals one.

There is still the possibility that we must search in all n directions, and that $(n \bmod p) \neq 0$. This is similar to the situation we face when $n < p$. We believe a careful use of the "speculative" function evaluation approach may help overcome this shortcoming. The "speculative" function evaluation approach to load balancing advocates the calculation of function values before it is clear that they will be required, so that they are already available should they be needed.

Our use of speculative function evaluation is based on the observation, first made in Chapter 3, that given an initial simplex we can predict a priori the points which the algorithm might visit — not only for the current iteration, but in subsequent iterations as well. We demonstrated this process in Figures 3.4–3.8. We must be careful to note, however, that for the sake of the convergence proof we assumed that there exists a lower bound on the length of the edges in the simplex when, in fact, no such bound exists. We can, nonetheless, predict in advance the points that will be visited in subsequent iterations. Now, however, the number of points in the grid is countably infinite, rather than finite, because we no longer have a fixed mesh size.

To insure load balancing when either $(n \bmod p) \neq 0$ or $n < p$, we anticipate the points that the algorithm is likely to visit next and use all "extra" processors to compute the function values at some subset of these points. We plan to develop strategies to decide which points should be considered first.

Two remarks should be made here. First, we do not need an explicit representation of the gradient either to determine a search direction or to decide if we have satisfied a *sufficient* decrease condition. The set of search directions is determined by the initial simplex and the algorithm only requires *simple* decrease in the function value at the best vertex. Thus, we do not require $O(n)$ function evaluations at each iteration before we can begin the search. Second, precisely because we can predict all the points the algorithm might consider, we can "start" the next iteration before we have actually finished the current iteration. Thus, we can develop strategies that use $O(p)$

function evaluations at each iteration, which means that we should be able to make better use of parallel machines.

The fact that the algorithm only requires simple decrease leads us to one final load balancing issue. Usually we assume that all function evaluation routines require the same amount of time to compute the function value. For the test problems we used in Chapter 5 this was certainly a reasonable assumption and thus made our measure of "effective" function evaluations valid. However, the problems we are ultimately interested in solving often involve iterative procedures — such as the solution of an ordinary differential equation. This means that the time required for each function evaluation could vary tremendously. However, since we only require simple decrease, which need not necessarily be the greatest decrease, as soon as we have found a replacement for v_0^k we can stop the computation and proceed immediately to the next iteration. Again, this should lead to better overall use of the available processing power.

6.1.2 Performance on non-differentiable and non-convex problems

We are interested in finding ways to handle problems that do not satisfy the assumptions required for our theoretical results. If we relax the differentiability assumption, the algorithm might get trapped in "bad" directions; i.e., it might not find a descent direction. When the function is not convex, the algorithm may still converge to a stationary point, but the stationary point may be a saddle point rather than a minimizer. In both cases, the method is hampered by the choice of the starting simplex. As we have made clear, the choice of the initial simplex will have a profound effect on the performance of the algorithm, since the search directions are chosen from a finite set of fixed search directions determined by the simplex used to start the procedure. Since some important problems are not differentiable or not convex, we would like to develop practical alternatives for overcoming an unfortunate choice of a starting simplex.

Differentiability clearly plays a key role. We have found simple examples for which the algorithm converges to a point that is not a local minimizer but at which the function is not differentiable. Assume the best vertex v_0^k is at a point where the function is not differentiable. Then, even if the n search directions are linearly independent we cannot be certain that at least one of the edges identifies a direction of descent. The search might then be "stuck" and the answer returned by the algorithm may be v_0^k. If v_0^k is not a minimizer, there are directions for which descent is guaranteed. The problem is that the algorithm did not identify one of these directions. The same sort of difficulty can occur when the function is not convex. We have constructed an example in which the best vertex v_0^k is a saddle point and none of the search directions lie in a direction of negative curvature. Again the search is stuck and the answer returned by the algorithm will be v_0^k, which is a stationary point but not a minimizer of the function. We would like to be able to recover in such instances by automatically generating a new simplex to give a new set of search directions. Either "sidestepping" or estimating curvature information may help generate new simplices, and thus new sets of search directions, when these difficulties are encountered.

Sidestepping

Our first idea involves "sidestepping" the difficulties mentioned above. Instead of halving the step lengths repeatedly, the algorithm could take a step in the $n-1$ dimensional (affine) subspace defined by the face of the simplex that includes all but the best vertex. The new face returned by this step would then replace the one in the original simplex. This would change the original shape of the simplex, but it would also give the procedure a new set of search directions. Sidestepping appears to preserve one of the important features that allows us to prove convergence for the multi-directional search algorithm. We would like to see if this extension can be included in our existing convergence theory. We would also like to see how well this idea performs in practice.

Estimating curvature information

Our second idea begins with the observation that, in some sense, the parallel algorithm implicitly approximates the gradient. While the algorithm does not explicitly compute finite difference approximations to the gradient, the algorithm instead considers a natural alternative: at every iteration it explores each direction in a set of n linearly independent search directions. The multi-directional search could then be viewed as an implicit finite difference approximation, in a different basis, to the gradient. The problem, particularly when dealing with functions that are not convex, is that the algorithm does not have any curvature information. To overcome this difficulty, we would like to approximate curvature information at the point returned by the algorithm. This issue has already been considered both by Spendley, Hext, and Himsworth [32] and by Nelder and Mead [24]. These researchers were interested in constructing the Hessian matrix at the minimizer in order to produce the variance-covariance matrix of the estimates. We are interested in pursuing this idea because we think we can use curvature information to construct automatic restart strategies. One possibility would be to use the Hessian to construct a new simplex with the edges oriented along the axes of a quadratic model of the function. This would determine a new set of search directions. Ideally, these search directions would better suit the particular problem.

Estimating curvature information would also allow us to explore a different application of the parallel algorithm. If we can furnish an approximation to the Hessian, we can start a Newton-like procedure. This suggests an interesting tie to work begin done on inverse problems by Williamson [37]: since the multi-directional search algorithm does not require a great deal of accuracy in the function evaluations, it could be used as the first stage in an approach to solving inverse problems. Once the multi-directional search has identified the neighborhood of a solution, the search could then switch to the Newton-like methods developed for inverse problems. The parallel direct search algorithm has the advantage of being relatively inexpensive but the Newton-like methods have fast local convergence properties. This scheme is designed to exploit both of these advantages.

6.2 Extending the theoretical results

6.2.1 Generalizations of the convergence theorem

The convergence theorem for the multi-directional search algorithm exploited two special features of the algorithm: that the algorithm searches over a finite set of fixed search direction and that the rescaling factors which determine the step sizes are fixed. There are several sequential direct search methods for which the same features hold, in particular, the pattern search algorithm of Hooke and Jeeves [17] and evolutionary operation, in its simplest form, as first proposed by Box [2].

We could view the simplex used in the multi-directional search algorithm as the "pattern" used to generate both the directions of search and the size of the steps. Other methods in this class simply use other "patterns" to determine the directions of search and the size of the steps, but the structure underlying these methods is the same. We thus believe we can develop a general convergence theory for this class of direct search methods, without modifying these algorithms. We know of no equivalent convergence theory for this class of direct search methods. We have recently become aware of a similar effort by Wen-ci [35], [36]. However, his convergence analysis requires a notion of sufficient decrease which it is not clear can be implemented in practice. His theory also cannot be extended to cover the multi-directional search algorithm because it requires the step size to be monotonically decreasing, which does not hold for the multi-directional search algorithm if expansion steps are allowed. On the other hand, it appears that our convergence theory can be extended to the variants Wen-ci proposes for both the pattern search algorithm of Hooke and Jeeves and the original simplex method of Spendley, Hext, and Himsworth. Céa [10] also gives a convergence result for a variant of the pattern search algorithm of Hooke and Jeeves, but he requires that the step sizes be strictly monotonically decreasing at every iteration — a modification we believe would be unsatisfactory in practice.

Finally, we believe that the convergence theorem can be extended to cover the non-differentiable case. Instead of convergence to a stationary point, we may be able to show convergence to a critical point, with few major modifications of the proof.

6.2.2 Exploring step size requirements

The convergence result we have for the multi-directional search algorithm raises an intriguing theoretical question. The step size rule of the parallel algorithm is reminiscent of the step size rule of the trust region methods. Both methods only require simple decrease on the value of the objective function; neither requires that the Armijo-Goldstein-Wolfe conditions be satisfied. Both methods employ a simple backtracking strategy to ensure that the step is not too short: steps are typically halved, but only until a point that produces decrease in the objective function value is found. Neither method safeguards against taking a step that is too long relative to the amount of decrease in the objective function — the other half of the Armijo-Goldstein-Wolfe conditions — and yet in neither case is this condition necessary to prove convergence. This is markedly different from the standard convergence results for line search methods. We would like to explore the relationship between the step size rule for the multi-directional search algorithm and the step size rule for the trust region methods to see what features they may share. The answer may shed new light on the way we view line searches.

Bibliography

[1] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[2] George E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, VI(2):81–101, June 1957.

[3] M. J. Box. A comparison of several current optimization methods, and the use of transformations in constrained problems. *The Computer Journal*, 9(1):67–77, May 1966.

[4] M. J. Box, D. Davies, and W. H. Swann. *Non-Linear Optimization Techniques*. ICI Monograph No. 5. Oliver & Boyd, Edinburgh, 1969.

[5] Gregory F. Brissey, Robert B. Spencer, and Charles L. Wilkins. High-speed algorithm for simplex optimization calculations. *Analytical Chemistry*, 51(13):2295–2297, November 1979.

[6] Samuel H. Brooks and M. Ray Mickey. Optimum estimation of gradient direction in steepest ascent experiments. *Biometrics*, 17:48–56, March 1961.

[7] K. W. C. Burton and G. Nickless. Optimisation via simplex: Part i. background, definitions and a simple application. *Chemometrics and Intelligent Laboratory Systems*, 1:135–149, 1987.

[8] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. Using parallel function evaulations to improve Hessian approximations for unconstrained optimization. Technical Report CS-CU-361-87, University of Colorado, 1987.

[9] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. Parallel quasi-Newton methods for unconstrained optimization. Technical Report CU-CS-396-88, University of Colorado, 1988.

[10] Jean Céa. *Optimisation : théorie et algorithmes.* Dunod, Paris, 1971.

[11] D. H. Chen, Z. Saleem, and D. W. Grace. A new simplex procedure for function minimization. *International Journal of Modelling & Simulation*, 6(3):81–85, 1986.

[12] Thomas F. Coleman and Guangye Li. Solving systems of nonlinear equations on a message-passing multiprocessor. Technical Report CORR 87-49, Faculty of Mathematics, University of Waterloo, December 1987.

[13] Stanley N. Deming and Stephen L. Morgan. Simplex optimization of variables in analytical chemistry. *Analytical Chemistry*, 45(3):278 A–283 A, March 1973.

[14] J. E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[15] J. E. Dennis, Jr. and D. J. Woods. Optimization on microcomputers: The Nelder-Mead simplex algorithm. In Arthur Wouk, editor, *New Computing Environments: Microcomputers in Large-Scale Computing*, pages 116–122. SIAM, Philadelphia, 1987.

[16] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, London, 1981.

[17] Robert Hooke and T. A. Jeeves. "Direct search" solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8(2):212–229, April 1961.

[18] S. L. S. Jacoby, J. S. Kowalik, and J. T. Pizzo. *Iterative Methods for Nonlinear Optimization Problems.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[19] Richard D. Krause and John A. Lott. Use of the simplex method to optimize analytical conditions in clinical chemistry. *Clinical Chemistry*, 20(7):775–782, 1974.

[20] Guangye Li and Thomas F. Coleman. A new method for solving triangular systems on a distributed memory message-passing multiprocessor. Technical Report CS-87-812, Computer Science Department, Cornell University, 1987.

[21] Guangye Li and Thomas F. Coleman. A parallel triangular solver for a distributed memory multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 9(3):485–502, May 1988.

[22] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[23] Stephen L. Morgan and Stanley N. Deming. Simplex optimization of analytical chemical methods. *Analytical Chemistry*, 46(9):1170–1181, August 1974.

[24] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[25] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[26] J. M. Parkinson and D. Hutchinson. An investigation into the efficiency of variants on the simplex method. In F. A. Lootsma, editor, *Numerical Methods for Non-linear Optimization*, pages 115–135. Academic Press, London and New York, 1972.

[27] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, July 1964.

[28] G. L. Ritter, S. R. Lowry, C. L. Wilkins, and T. L. Isenhour. Simplex pattern recognition. *Analytical Chemistry*, 47(12):1951–1956, October 1975.

[29] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, October 1960.

[30] M. W. Routh, P. A. Swartz, and M. B. Denton. Performance of the super modified simplex. *Analytical Chemistry*, 49(9):1422–1428, August 1977.

[31] R. B. Schnabel. Concurrent function evaluations in local and global optimization. *Computer Methods in Applied Mechanics and Engineering*, 64:537–552, 1987.

[32] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, November 1962.

[33] W. H. Swann. Direct search methods. In W. Murray, editor, *Numerical Methods for Unconstrained Optimization*, pages 13–28. Academic Press, London and New York, 1972.

[34] D. A. Walmsley. The simplex method for minimisation of a general function. Supplementary Report 686, Assessment Division, Transport Systems Department, Transport and Road Research Laboratory, Crowthorne, Berkshire, 1981.

[35] Yu Wen-ci. The convergence property of the simplex evolutionary techniques. *Scientia Sinica*, Special Issue of Mathematics(1), 1979.

[36] Yu Wen-ci. Positive basis and a class of direct search techniques. *Scientia Sinica*, Special Issue of Mathematics(1), 1979.

[37] K. A. Williamson. *Parameter Identification in Systems of Ordinary Differential Equations*. PhD thesis, Rice University, Houston, Texas, 1990. In preparation.

[38] Daniel J. Woods. *An Interactive Approach for Solving Multi-Objective Optimization Problems*. PhD thesis, Rice University, 1985.

[39] Willard I. Zangwill. Minimizing a function without calculating derivatives. *The Computer Journal*, 10(3):293–296, November 1967.