DISSERTATION

EXPLORING THE BIAS OF DIRECT SEARCH AND EVOLUTIONARY OPTIMIZATION

Submitted by
Monte Lunacek
Department of Computer Science

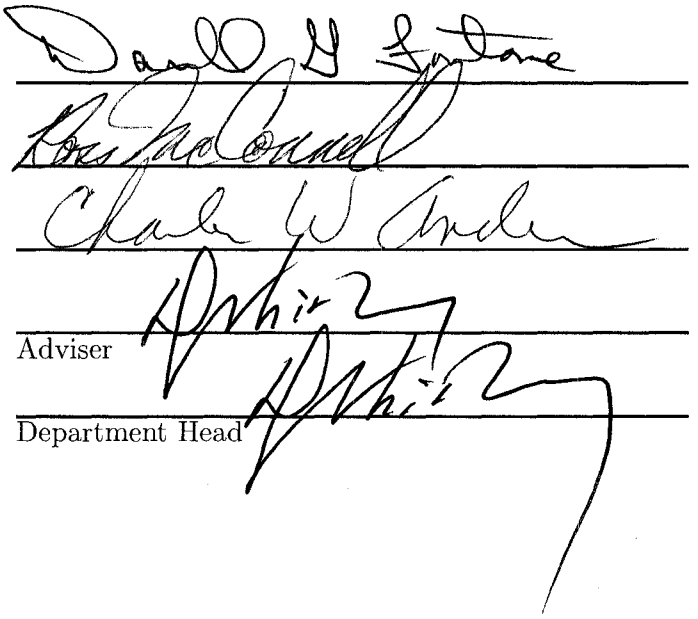
In partial fulfillment of the requirements
for the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Summer 2008

COLORADO STATE UNIVERSITY

May 9, 2008

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY MONTE LUNACEK ENTITLED EXPLORING THE BIAS OF DIRECT SEARCH AND EVOLUTIONARY OPTIMIZATION BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work



Adviser

Department Head

ABSTRACT OF DISSERTATION

EXPLORING THE BIAS OF DIRECT SEARCH AND EVOLUTIONARY OPTIMIZATION

There are many applications in science that yield the following optimization problem: given an objective function, which set of input decision variables produce the largest or smallest result? Optimization algorithms attempt to answer this question by searching for competitive solutions within an application's domain. But every search algorithm has some particular bias. Our results show that search algorithms are more effective when they cope with the features that make a particular application difficult.

Evolutionary algorithms are stochastic population-based search methods that are often designed to perform well on problems containing many local optima. Although this is a critical feature, the number of local optima in the search space is not necessarily indicative of problem difficulty. The objective of this dissertation is to investigate how two relatively unexplored problem features, ridges and global structure, impact the performance of evolutionary parameter optimization. We show that problems containing these features can cause evolutionary algorithms to fail in unexpected ways.

For example, the condition number of a problem is one way to quantify a ridge feature. When a simple unimodal surface has a high condition number, we show that the resulting narrow ridge can make many evolutionary algorithms extremely inefficient. Some even fail.

Similarly, funnels are one way of categorizing a problem's global structure. A single-funnel problem is one where the local optima are clustered together such that there exists a global trend toward the best solution. This trend is less predicable on problems that contain multiple funnels. We describe a metric that distinguishes problems based on this characteristic. Then we show that the global structure of the problem can render successful global search strategies ineffective on relatively simple multi-modal surfaces. Our proposed strategy that performs well on problems with

multiple funnels is counter-intuitive.

These issues impact two real-world applications: an atmospheric science inversion model and a configurational chemistry problem. We find that exploiting ridges and global structure results in more effective solutions on these difficult real-world problems. This adds integrity to our perspective on how problem features interact with search algorithms, and more clearly exposes the bias of direct search and evolutionary algorithms.

Monte Lunacek
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 2008

ACKNOWLEDGMENTS

I am extremely grateful for all the support and guidance I received during my graduate career. First, I would like to thank my advisor Darrell Whitley. His encouragement kept me going strong and gave me the confidence I needed during the times when I was struggling. I especially appreciate the long hours he invested in me and the knowledge and wisdom he shared. I feel fortunate to have had the opportunity to work with one of the most respected scientists in evolutionary computation. I appreciate the freedom he gave me to explore topics I was interested in, as well as the advise and direction he offered. I can't image a better experience. The primary reason I have finished this Ph.D. is because of the strong relationship I have with Dr. Whitley. I am grateful for his mentoring and friendship. Thank you so much Darrell.

I feel fortunate to have met so many wonderful people while living in Fort Collins. In particular, to my friends in the "fun zone", I am deeply grateful for your support, encouragement, and all the laughter and fun.

My parents, Ted and Nancy, are amazing people. I appreciate all they have done for me during graduate school and the support they provide. Their continual encouragement and support, friendship, and love is the foundational in everything I do.

I appreciate the comments and feedback from my committee members, Charles Anderson, Ross McConnell, and Darrell Fontane. Thank you investing your time and making my dissertation a better paper.

The research in this dissertation is a product of collaboration with several people. I appreciate the perspective, support, advise, and ideas that Jean-paul Watson and William Hart at Sandia National Labs offered. Jean-Paul first introduced me to the Lennard-Jones clusters and Bill pushed me to look more closely at self-adaptation in evolution strategies. I enjoyed working with Philip Gabriel and Graeme Stephens on their temperature problem and am grateful for all the time they in-

vested. I am especially grateful for the support of other graduate students in the Computer Science department. In particular, I worked most closely with James Knight, Andrew Sutton, and Artem Sokolov. Their ideas and comments were invaluable.

This work was supported by the National Science Foundation under Grant No.0117209 and by Sandia National Labs. The research presented in this dissertation is based on the following papers:

Chapter 3

- *Comparing the Niches of CMA-ES, CHC, and Pattern Search Using Diverse Benchmarks* with Darrell Whitley and Artem Sokolov [106].
- *Alternative Implementations of the Griewank Function* with Artem Sokolov and Darrell Whitley [91].

Chapter 4

- *Ruffled by Ridges: How Evolutionary Algorithms Can Fail* with Darrell Whitley and James Knight [105].
- *Searching for Balance: Understanding Self-adaptation of Ridge Functions* with Darrell Whitley [57].

Chapter 5

- *Applying Search Algorithms to the Temperature Inversion Problem* with Darrell Whitley, Philip Gabriel, and Graeme Stephens [58].

Chapter 6

- *Measuring Mobility and the Performance of Global Search Algorithms* with Darrell Whitley and James Knight [59].
- *Function dispersion and the CMA Evolution Strategy* with Darrell Whitley [56].

Chapter 7

- *The Impact of Global Structure on Search* (submitted April 2008) with Darrell Whitley, Andrew Sutton, and Jean-paul Watson [60].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Many applications in science and engineering that involve a set of input variables. Each application can be viewed as a *black box* containing several "knobs" that represent specific input parameters. The optimization problem is to find the set of knob positions that returns the largest or smallest value from the given application. If the black box represents an objective function, $f(\vec{x})$, then *parameter optimization* is the task of finding the set of parameter values, denoted $\vec{x}$, that yields the greatest or least objective function value. The best set of parameter values is called the *global optimum.*

Search algorithms are used to locate globally competitive solutions within the domain of a given parameter optimization problem. Some search algorithms, like classic *gradient-based* methods, use derivative information from the objective function to guide search. *Direct-search* methods use only the values returned by the objective function as a means of finding better candidate solutions. These strategies are also referred to as *derivative-free* methods, and can be either stochastic or deterministic. *Stochastic* algorithms have some form of random variation built into their search heuristics. *Genetic algorithms* [45; 24] and *evolution strategies* [88] are two well-known direct-search stochastic algorithms. Algorithms that use a deterministic search heuristic include the popular *Nelder-Mead simplex* algorithm [69] and the general class of *pattern search* methods [96], which includes *steepest-descent local search.*

All search algorithms have some particular bias: since there are no general purpose search methods that work well on all problems, every search algorithm must have a specific domain for which it is likely to perform the best. But rarely do the developers of new heuristic search methods document the types of problems on which their algorithm is likely to yield good performance. While there are exceptions to this observation, the relative strengths and limitations of direct search

methods are not well understood, in part, because algorithm behavior is often inferred through comparative studies that are greatly influenced by the *selection of algorithms* compared and the *problems* used to evaluate them.

Given the complexity of both *algorithms* and *objective functions*, most researchers realize that empirical comparisons are necessary for understanding the relative behavior of search algorithms [13]. New search algorithms are often shown to be either more efficient or effective on a given benchmark problem when compared to other established methods. But often, search algorithms are developed and evaluated in isolation from other algorithms that would also make good candidates for the problems that they are trying to solve. Sometimes, new heuristic search methods are compared against an established, but ineffective, canonical strategy rather than other functional algorithms. The problem is that there may exist a more efficient or effective way of solving the particular problem. As a result, there are inherent limitations in any comparative study; since it is impractical to compare all algorithms across all problems, understanding the strengths and limitations of any single algorithm, in the general case, is intractable.

Artificial functions are normally used in empirical studies because real applications are often too computationally costly and acquiring (or disseminating) them is difficult. Ideally, these synthetic test problems embody the characteristics of real applications. The assumption is that if an algorithm performs well on the artificial test functions, then it will also work well on other applications. This assumption fails when realistic problem characteristic are either not present, or have a limited representation in the overall test suites. Consequently, building *challenging*, *diverse*, and *realistic* test functions implies understanding the characteristics that make realistic parameter optimization problems difficult. The value of any comparative study is also limited, in part, by our ability to understand the characteristics of realistic problems, and by our ability to embed these features into benchmark test problems.

1.1 Problems Features

Several features have been identified that help explain why some optimization problems appear more challenging than others when compared against a varying assortment of algorithms. These features include *dimensionality*, *noise*, *symmetry*, *separability*, *modality*, *scale*, and *global structure*. It is

well accepted that increasing the number of objective parameters, or *dimensionality*, usually makes an optimization problem more difficult to solve. *Noisy* objective functions, which are common in many real-world applications, can make an otherwise smooth landscape appear more rugged from the perspective of search. Adding even a small amount of noise to an objective function can cause some direct search strategies to fail [3]. Highly *symmetric* functions can be easier for some algorithms to solve than problems that have a more asymmetry structure [109].

Whitley *et al.* [107; 109] noticed that most of the test functions used to evaluate evolutionary algorithms are *separable*. Separable problems contain no non-linear interactions between the parameters of the objective function—an unrealistic assumption to make about most real-world optimization problems. A separable objective function can easily be solved by searching for the optimal solution separately in each dimension [109]. The way in which problem *separability* affects performance has been extensively studied [83; 43; 71; 31; 78].

The number of local optima, or *modality*, can also affect problem difficulty. Often an algorithm fails to locate the global optimal because it has converged to a local optima. One common assumption is that highly multimodal problems are more difficult than those with fewer local optima because there are more "traps" where search can get stuck. Figure 1.1 displays two views of a frequently used *multimodal* benchmark test function. The surface on the left is the landscape generated from a two-dimensional instance of the Rastrigin function. Each parameter translates to a single dimension and the objective function value determines the elevation of the landscape. The one-dimensional function on the right is a slice of a single dimension. Each local optima creates a barrier for search.

Problems that are *poorly scaled* have a different rate of change between the parameters of the objective function. This creates a long narrow *ridge* in the landscape that can greatly impact the local efficiency of search. The ridge axis corresponds to the direction of the smallest rate of change in the objective function. Figure 1.2 shows two contour plots that give a birds-eye view of the landscape. The right-most surface has a more dramatic difference in scale because the objective function has been "squeezed" in the direction of the ridge axis. Although this surface is smooth and unimodal, ridges can cause a wide variety of algorithms to behave inefficiently, including line search and gradient descent [81], pattern search methods [54], and evolutionary algorithms [105].

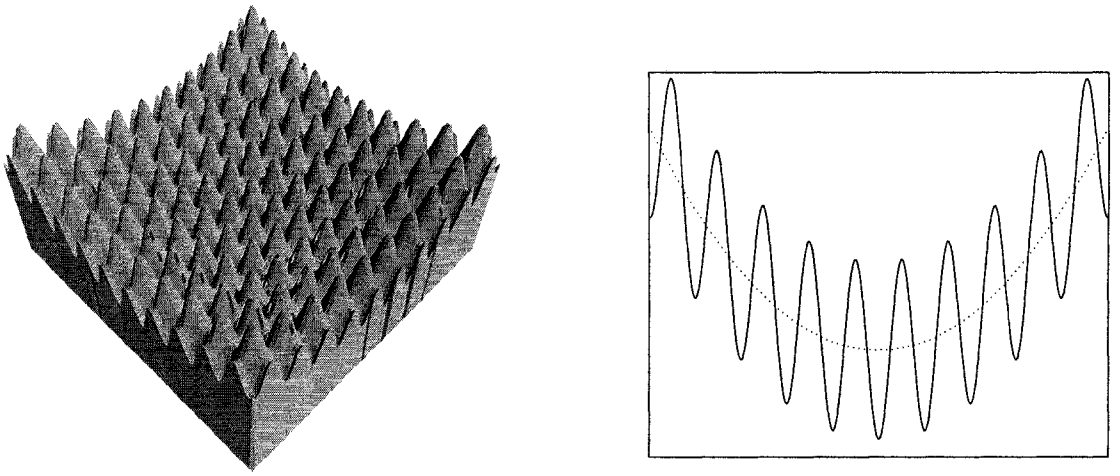


Figure 1.1: The multimodal Rastrigin test function. The surface on the left is the objective function landscape of a two-dimensional problem instance. The right function is a slice of a single dimension of the landscape on the left. Each local optima can act as a "trap" for search.

Hu *et al.* were probably the first to document that some multimodal objective functions appeared to have a convex *global structure* [48]. They comment that the local optima of such functions can be viewed as "blips" of a more predicable underlying convex surface. Addis *et al.* complements this by suggesting that some multimodal surfaces can be seen as perturbations of a simple underlying structure that has a low number of local optima [2]. Referring back to Figure 1.1, the dashed curve on the one-dimensional slice of Rastrigin's function highlights this problem's unimodal *global structure.*

A convex *global structure* may be described in different ways. Several optimization problems in computational chemistry exhibit "single funnel" behavior. Although the surface is multimodal, the local optima of a *single funnel* landscape are clustered such that search is "funneled" down towards the global optimum. A "funnel" essentially refers to a part of an underlying global structure that is convex. In this way, Rastrigin's *global structure* can be described as having a single funnel. Globally convex structures were also discovered in several combinatorial optimization problems, including the well-know Traveling Salesperson Problem (TSP). Boese, Kahng, and Muddu [16] discovered a strong correlation between the fitness of a given solution and its distance to the global optima. Problems of this type exhibit a "big valley" structure; the best local optima are close to the global optima and more distant local optima have poorer fitness values.

Figure 1.3 shows the correlation between objective function cost (y-axis) and distance to the

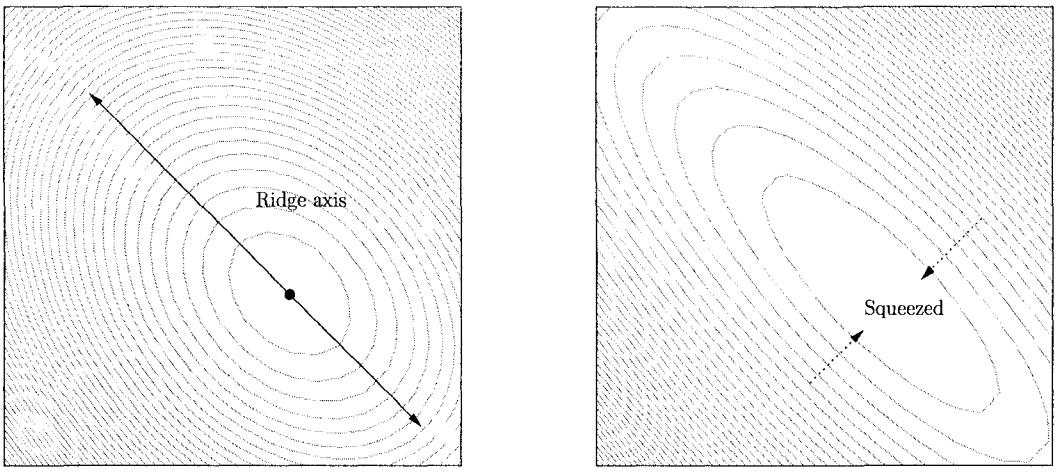


Figure 1.2: The contour plots of two ridge functions. The left-most figure shows the ridge axis of a simple two-dimensional, non-separable *ridge* function. The direction of smallest change in the objective function corresponds to the ridge axis. The right-most figure shows a more exaggerated ridge, where the sides of walls of ridge have been "squeezed" together.

global optima (x-axis) for two domains where big valley topology has been observed: a discrete Traveling Salesperson Problem and a continuous configuration chemistry problem. The left-most graph represents 2,500 local optima from a benchmark TSP problem [16]. The local optima in the right figure are taken from a 19−atom Lennard-Jones instance [26] , an optimization problem which we discuss below. The details of each problem are not important here. In both applications, however, an improvement in objective function cost (y-axis) implies that the local optima is generally closer to the global optima (x-axis).

1.2 Objective

The evolutionary algorithm community has not developed a clear understanding of how search algorithms interact with different types of problem features. Most new research is still focused on escaping local optima (modality), and maybe on rotational invariance (separability), but there is very little discussion of how some of the other features we just described impact search. Yet we strongly believe that if an algorithm is going to be successful on a particular application, its heuristics must address the features that exist with in the objective function.

The objective of this dissertation is to investigate how ridges and global structure impact the performance of evolutionary algorithms.

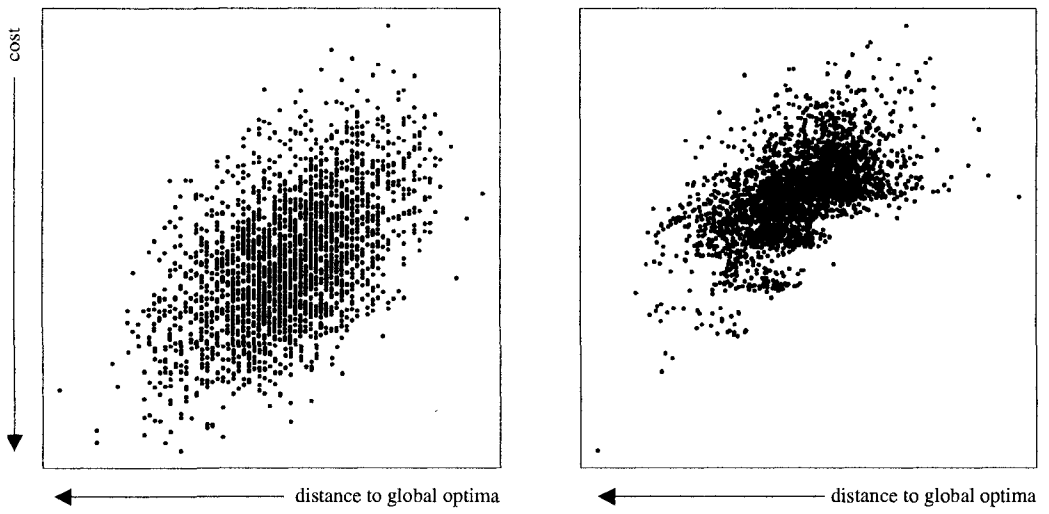


Figure 1.3: Big valley landscapes have been noticed in discrete Traveling Salesperson Problems and several continuous Lennard Jones cluster instances. The left-most graph was reconstructed from TSP data collected by Boese, Kahng, and Muddu [16]. The right-most graph was reconstructed from local optima based on the 19-atom Lennard-Jones cluster generated by Doye, Miller, and Wales [26]. The y-axis represents the objective function cost and the x-axis measures distance from the global optima. In both cases, there is a general trend where a decrease in objective function cost (y-axis) tends to imply the local optima is closer to the global solution (x-axis).

This objective was cultivated through several observations. First, we have been working on an application that computes the inverse of a weather prediction model [58]. Sometimes first-order derivatives can be calculated analytically for this application, but in the more complex and realistic models, the analytical gradient is impossible to compute [30]. It would appear that this problem is a perfect candidate for direct search methods. However, our results indicate that many well known evolutionary algorithms and local search methods do not yield acceptable solutions. This is because there are *ridges* in the search space. Although the "ridge problem" has been acknowledged and documented in mathematical literature on derivative-free optimization, there is surprisingly very little discussion of this feature from an evolutionary optimization perspective.

Second, not all optimization problems have a *big valley* structure. Sometimes very distant solutions can be nearly as good as the global optima. There are several applications in computational biology that are considered to be difficult because the objective function appears to have multiple local optima that form distinct, spatially separate clusters in the search space [99]. Lennard-Jones clusters, for instance, are a class of configuration optimization problems where the goal is to find the spatial position for a set of atoms that has the smallest potential energy. While some instances

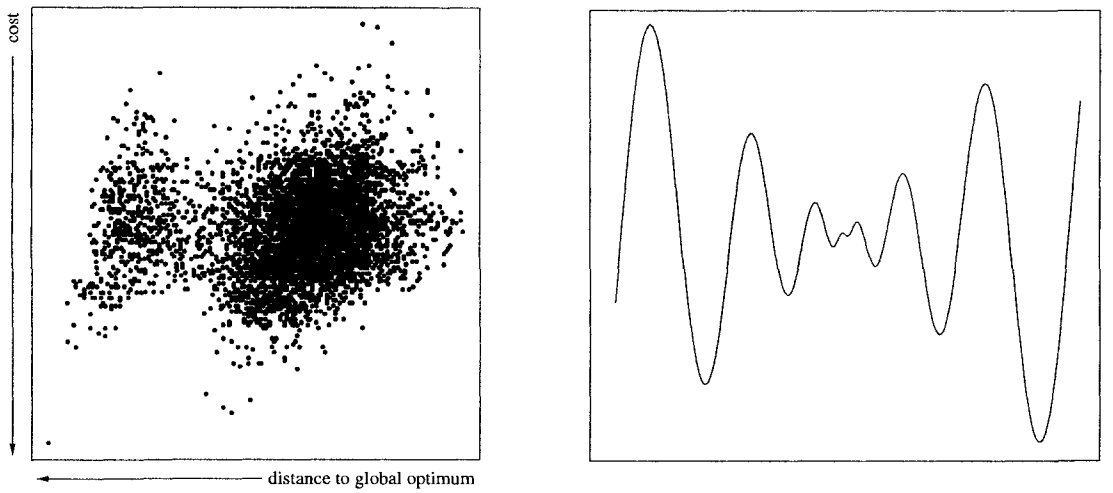


Figure 1.4: Not all realistic applications have a "big valley" global structure. The left graph is a reconstruction of data obtained from Doye, Miller, and Wales [26] based on the 38-atom Lennard-Jones cluster. The y-axis represents the objective function cost and the x-axis measures distance from the global optima. Notice that objective function cost (y-axis) is not tightly correlated with distance to the global solution (x-axis). The Schwefel test function (right) is an example of an artificial test problem that does not have a unimodal global structure.

of this problem do have a single funnel (see Figure 1.3), there are several instances that *do not* have this type of convex global structure. Researchers are starting to observe that it is the underlying problem structure that makes multimodal problems easier or more difficult, not the actual number of local optima [77]. As a result, the underlying structure of a problem can have a significant impact on how difficult it will be to find the global optimum.

Figure 1.4 shows an example of two functions that do not have a convex global structure. The left graph is a reconstruction the 38-atom Lennard-Jones cluster fitness-distance plot created by Doye, Miller, and Wales [26]. Notice the two "funnels" that form in this instance. The right plot is a one-dimensional slice of the Schwefel test function. Notice that this function does not have a *big valley* structure. The second most competitive solution is far from the global optima, which means that an average decrease in fitness does not necessarily imply that search is getting closer to the best solution.

Despite the impact *global structure* has on algorithm performance, the evolutionary algorithm community has largely ignored how this feature plays an important role in global optimization. Instead, multimodal problems are treated as if they are indistinguishable even though the actual problem difficulty can vary wildly from easy to impossible. One consequence of ignoring global

structure in an empirical study is that some algorithms are claimed to be effective global optimization strategies based how well they solve unrealistic and potentially easy multimodal functions.

1.3 Approach

Like most empirical studies, the conclusions we observe have some inherent limitations because, as mentioned, it is infeasible to identify and compare the very best algorithms against a broad and diverse set of problems. Therefore, our empirical conclusions are still shadowed by two reasonable complaints: First, algorithms are often evaluated in isolation from other strong strategies, including optimization algorithms that are being developed outside the evolutionary algorithm community. Second, the test problems are too easy and do not resemble realistic applications.

We address the first concern by carefully selecting algorithms with diverse heuristics. In the next chapter, we describe several search algorithms, both direct search and those that use gradients. We include simple direct search methods for perspective, like the *self-adaptive evolution strategy* and *steepest descent local search.* We also describe more complex evolutionary algorithms such as CHC [32], a non-traditional genetic algorithm, and an evolution strategy using Covariance Matrix Adaptation, or CMA-ES [43].

While these algorithms share some common features, they are also unique in distinct ways. For example, most evolutionary algorithms use a set of potential solutions that is initialized in a disperse way in order to explore the search space. Over time, variation and selection drive this set of candidate solutions, or *population*, towards more concise regions of the landscape. The exact nature of this process—variation and then selection—is one characteristic that makes an algorithm unique. We explain and highlight these differences in greater detail in Chapter 2, but point out here that CMA-ES, for example, relies heavily on a so-called *mutation* operator to create variation, while CHC exclusively uses what is known as *crossover* to vary population members from one iteration to the next. Algorithms also differ in how new populations are chosen. CHC requires that the new population members compete with the current population for future reproductive opportunity. CMA-ES, and some forms of self-adaptation, considers only the offspring of the current generation as potential candidates for the next iteration.

These subtle differences create a diverse group of heuristic strategies. We also include recent

pattern search algorithms in some of our comparisons in order to give a perspective on what is happening outside the evolutionary algorithm community. In contrast to the stochastic nature of most evolutionary algorithms, some of the methods included in this dissertation are largely deterministic once they have been initialized. In Chapter 5 we compare evolutionary algorithms with gradient-based algorithms. This gives a good perspective on how evolutionary search compares with gradient-based algorithms on ridges functions. Finally, we also compare specialized algorithms where appropriate. This includes algorithms that exploit physical continuity (Chapter 5) and algorithms designed to exploit funnel structure (Chapter 7).

Throughout this dissertation, difficult test functions and realistic applications are used in order to make our test suite more challenging and complex. As with our algorithm selection, these functions are not randomly chosen. We deliberately select them based on the features that make them difficult. In Chapter 3, we introduce several of the standard artificial test functions used to evaluate evolutionary algorithm and discuss the features that are believed to make these problems difficult. Since some algorithms exploit separability, we create non-separable functions using rotation [83]. One popular function actually gets easier in high dimensions. We provide some alternative implementations that makes the difficulty of this problem scale well as dimensionality increases. We also introduce a class of multi-funnel test functions in Chapter 7.

Realistic applications also add to the diversity and difficulty of our empirical tests. We introduce a synthetic version of a geophysics problem in Chapter 3. Later, in Chapter 5, we compare several search strategies on the previously mentioned atmospheric science problem. We also present results in Chapter 7 of several algorithms applied to some Lennard-Jones cluster instances.

With a thoughtful selection of algorithms and functions, we mitigate some of the restrictions that inherently come with an empirical study. Then, in Chapters 3—7, we explore how ridges and global structure impact search from several perspectives. The following subsections summarize the objectives of each chapter.

1.3.1 Comparing Algorithms

When we compare some of these diverse algorithms on a more challenging set of test functions in Chapter 3, we find that there is no single algorithm that works well on every problem. We find that

observable differences in algorithm performance can be related to problem features. In particular, most of the discrepancies that we witness in algorithm performance can be explained by either separability, ridges, or global structure.

The way that separability impacts search has been widely explored, and we review this related literature in Chapter 3. There is significantly less research devoted to *ridges* and *global structure*. Yet they account for some of the discrepancies we notice. We assert that these kinds of differences are rarely reported in optimization literature, especially how they relate to problem characteristics. Chapter 3 brings to light the fact that the evolutionary optimization community still has not developed a clear understanding of how different algorithms exploit different types of problem features.

1.3.2 Ridges

Ridges in the search space can make some algorithms extremely inefficient. The ridge feature is interesting from a search perspective because it affects different algorithms in different ways.

In Chapter 4 we explain why algorithms that are biased in search direction will behave inefficiently or fail on ridges when their search direction is not aligned with the ridge axis. In particular, we show that a simple local search algorithm, that uses a neighborhood search pattern aligned with the coordinate axis, creates false local optima in the search space. We also look at how a rotated representation of the search coordinates can make this simple local search algorithm dramatically more efficient.

We also explain why ridge functions are equally difficult for algorithms that sample points in an unbiased direction. We provide empirical evidence that the self-adaptive evolution strategy, which we discuss in Chapter 2, will decrease its step-size until the selection of the next iterations step-size is not biased toward larger or smaller step-size values.

1.3.3 Temperature Inversion

We present a temperature inverse problem in Chapter 5, which is a *forward model* that relates vertical temperature profiles to observed measurements. What is actually needed is the inverse: given a set of observations, what is the corresponding temperature profile? These profiles are used in global atmospheric circulation and weather prediction models, and must be found efficiently.

We formally show that several of the evolutionary algorithms we describe in chapter 2 fail to

find useful solutions to this application. Furthermore, the time required by the *only* algorithm that does find useful profiles is too long for practical use. Ridges in the search space account for part of the problem difficultly. There is also a high degree on nonlinearity between the input parameters and a strong bias toward certain regions of the search space. We discuss how this relates to our results.

The results we present are unsatisfying from a practical point of view because we still have not identified an algorithm that is efficient and effective. At the same time, the heuristic search methods we are applying are used "off the shelf" and do not make any assumptions about the search space. Usually incorporating some problem-specific knowledge will increase performance.

The temperature problem has two features that we exploit. First, spatially close object parameters should have similar temperature values. This is because we don't expect there to be a sharp drop in temperature as we ascend into a relatively stable atmosphere. We examine two methods that exploit the smooth nature of the temperature profile; Salomon's *optimize and refine* algorithm **??** and a new algorithm we call *tube search*. Both find relatively useful (and smooth) temperature profiles using a reasonable number of calls to the evaluation function. Second, the temperature problem actually belongs to a family of *nonlinear least-squares* search problems. The structure of the objective function is exploited by the gradient-based Levenberg-Marquardt algorithm. Compared with other gradient methods and evolutionary algorithms, this method is extremely fast.

The temperature problem adds integrity to our perspective on ridges. It serves as a real application that identifies which algorithms address the ridge problem and which do not. Our investigation indicates that evolutionary algorithms, in general, do not focus on ridges in the search space.

1.3.4 Global Structure: Measuring Function and Algorithm Dispersion

Although most of the test functions we describe in Chapter 3 have a "big valley" structure, others do not. Our results suggest that problems with a more complex underlying structure are more difficult than those with a more predictable and simple global structure. This is consistent with other related research [77]. Yet often these problems are treated as if they belong to the same class of "multimodal" test functions.

In Chapter 6, we explore how global structure impacts search. We start this chapter by introduc-

ing *function dispersion*, an algorithm independent metric that allows us to categorize the benchmark test functions based on their underlying global structure. A *low dispersion* function is one that has a "single funnel", or "big valley". *High dispersion* is the opposite. The best regions of the search space in these functions are not clustered together, but tend to be spread out and *disperse*.

On high dispersion functions—those that lack a predictable "big valley" effect—we reason that the most effective algorithms will ultimately compare the best local optima in the search space. This means that the local optima they visit are likely to be very disperse. To measure this, we define a new metric, *mobility*, that measures *algorithm dispersion*, which we define to be the dispersion of the best local optima visited by search. We show that algorithms that visit more disperse local optima (higher mobility) tend to find better overall solutions on high dispersion functions. Unfortunately, we are unable to capture this trend in dimensions higher than about 10. This is partly because the algorithms become less variable in terms of performance as dimensionality increases.

The CMA-ES algorithm, which is driven by local search heuristics, must rely on restarts to increase its mobility. We explore how function dispersion impacts CMA-ES and show that our dispersion metric can predict how well CMA-ES will perform on multimodal functions. Our results indicate that CMA-ES works very well on low dispersion functions, but has difficulties when compared with high dispersion functions. We discuss why CMA-ES is less efficient on problems where the best regions of the search space are too spread out.

1.3.5 When does Global Structure Matter?

We continue investigating global structure in Chapter 7 by formally describing the Lennard-Jones cluster optimization problem and discussing what researchers believe make multi-funnel instances of this problem difficult for search. Then we show that evolutionary algorithm do not perform well on applications where the optimal solution belongs to a funnel that occupies a small proportion of the search space. We found that part of the problem is the way that evolutionary algorithms often initialize their populations.

It is commonly believed that population-based methods are better at optimizing multimodal functions because they tend to explore more of the fitness landscape before their population converges to a compact, globally competitive region. The assumption here is that by *exploring* the

search space first—and then *exploiting* promising regions next—population-based algorithms have a better perspective of the entire search space than *local search* methods. But we show that when the optimal solution lies at the bottom of a proportionally small funnel, too much exploration can cause search to fail.

We isolate this feature by introducing a class of double-funnel test functions where the number of funnels remains constant (two) as dimensionality increases and the relative size and depth of each funnel is adjustable. Then, we show that evolutionary algorithms utilizing the "explore then exploit" philosophy are increasingly less effective on some of our double-funnel test functions. Our results indicate that too much exploration hinders search in high dimensions. When we reduced the amount of exploration in a simple evolution strategy by using a variable population size, we find this results in a performance gain on our most difficult double-funnel functions. One surprising result from chapter 7 is that limiting the degree to which an algorithm explores the search space can actually improve its global search performance.

The Lennard-Jones clusters demonstrate that a less predictable underlying global structure is a realistic feature that can be difficult for search. We find that exploration is biased towards large funnels, and that this limits the ability of the algorithms we compare to find optimal solutions that exist in proportionally smaller funnels.

1.4 Summary

There is no one algorithm that will behave efficiently and effectively on every type of optimization problem. This realization is widely accepted within the majority of the optimization community and is evidenced in both theory and by practitioners, who continually incorporate problem-specific knowledge into a particular algorithm in order to find more effective solutions, more efficiently. With this in mind, a fruitful direction of research is to identify where algorithms perform the best, and where they fail. Identifying new features that make real problems difficult for search is a required first step. Understanding how different algorithms interact with these features is the next logical consequence. We are exploring how ridges and global structure impact evolutionary algorithms and in the process, we are better understanding the bias of evolutionary parameter optimization.

The evolutionary algorithm community has still not completely embraced this notion of feature-

based optimization development. Some algorithms, like CMA-ES, are developed with certain features in mind, such as separability and scale. But this seems to be a rare exception in algorithm development. It is often still the case that new algorithms are introduced as a metaphor first, and then claimed to do well on a potentially easy test suite. The problem is, we have failed to learn anything about search in this process.

A potentially more productive extension of adopting the idea that we are not looking for a "one size fits all" algorithm is to design heuristics that directly address certain problem features and identify those function characteristics that cause them to fail. This will ultimately create more diverse heuristic methods and aid our understanding of how algorithms interact with parameter optimization landscape features.

Ridges and *global structure* are two such features. They challenge the common belief that unimodal problems are easy and multimodal problems are hard. For example, *ridges* can make unimodal surfaces more challenging for search than some multimodal functions that have a convex *global structure*. At the same time, the *global structure* of a problem may be more indicative of its problem difficulty than the *modality*.

The research presented here ultimately shows that evolutionary algorithms have, for the most part, ignored how these features impact search. Paying attention to these features—and implementing strategies that address them—may expand the role that evolutionary algorithms play within the global optimization community.

Chapter 2

Search Algorithms

Prior to the 1950's, the most popular methods for solving unconstrained optimization problems were *gradient-based* techniques. These methods, which require derivative objective function information, include well-known strategies such as *steepest-descent*, the *conjugate gradient* method, and the general class of *quasi-Newton* methods.

Researchers eventually began exploring new ways to solve optimization problems without the use of derivatives. Davidon describes an early method of a basic *pattern search* where the features were varied "one parameter at a time by steps of the same magnitude", decreasing the step size when no improvement could be found [22]. A few years later, in 1966, Fogel *et al.* describe their optimization process of finite state machines as *evolutionary programming* [36]. Within the next decade, John Holland began developing a strategy based on selecting, recombining, and mutating the features of candidate solutions [45]. This created a class of heuristic techniques later referred to as *genetic algorithms* [24]. Around the same time, Ingo Rechenberg independently proposed the idea of random changes to the parameter values of a difficult flow engineering problem—an idea he conceived from nature's own mutations [88]. Techniques based on this foundation are known as *evolution strategies.*

Following these early developments, successful variations and modifications have kept heuristic search methods extremely popular in practice. This chapter describes several search algorithms that are suitable for solving unconstrained nonlinear optimization problems. Specifically, we discuss stochastic direct search algorithms, including a discrete genetic algorithm and two evolution strategies. We also look at deterministic methods, both pattern-search and gradient-based.

2.1 Problem Representation

We are interested in objective functions that quantify a set of real-valued input values. The minimization problem is to find the input vector that produces the smallest value. Formally, we are interested in minimizing an unconstrained nonlinear optimization function that maps a set of real-valued input parameters to a single value:

$$f : \mathbb{R}^N \rightarrow \mathbb{R}$$

Most of the algorithms described in this chapter represent candidate solutions as real-valued vectors. Therefore, the individual candidate solutions can be directly applied to the objective function. However, some of the algorithms described in this chapter use heuristics that require a discrete representation. In particular, *CHC*, a genetic algorithm, and *local search*, a deterministic pattern search algorithm, both use finite bit-strings to represent candidate solutions, and create new candidate solutions using heuristics that operate on these bit-strings. Instead of having the full range of machine precision available in the continuous domain, they often operate on candidate solutions that have a lower, less precise, range of values for each input parameter.

Applying these discrete algorithms to continuous parameter optimization problems requires a unique mapping from the discrete representation (e.g. the bit-string) to a real-valued representation before the objective function can be evaluated. A *Standard Binary* encoding scheme with a predetermined level of precision is one way to do this. Traditionally, 10- to 20-bits of precision is used for each parameter. This means that there are 2^{10} or 2^{20} values that a particular input parameter can hold. In the following example, the vector $\vec{y}$ is a three dimensional 5-bit representation of a candidate solution. The corresponding vector $\vec{x}$ is the Standard Binary transformation of $\vec{y}$ that can be applied to the objective function (assuming a domain of $[0, 2^5]$).

$$\vec{y} = [01010, 11110, 00101]^T \quad \text{and} \quad \vec{x} = [10, 30, 5]^T$$

Gray codes are an alternative to Standard Binary representation. Here, Gray code refers to the *Standard Binary Reflected Gray* code. One reason Gray codes are popular is because they remove *Hamming Cliffs* found in binary encoding schemes. A Hamming Cliff occurs when two adjacent numbers in numeric space have complementary bit representation. In terms of Hamming Distance,

which measures the number of bit positions that are different between two bit strings, a Hamming Cliff occurs when two strings maximally distant from each other in Hamming space. For example, 7 and 8 are adjacent neighbors in integer space, but their 4-bit binary representations, 0111 and 1000, are complements of each other in Standard Binary. They have a Hamming Distance 4.

Gray codes also preserve the adjacency found in numeric representation, because by definition, adjacent integers are adjacent neighbors. In other words, each neighbor under Gray code representation is only Hamming Distance 1 from adjacent neighbors in integer representation. In the previous example, 7 and 8 are Hamming Distance 4 from each other under Standard Binary. But under Standard Binary Reflected Gray code, their representations are 0100 and 1100, which are only Hamming Distance 1 from each other.

The consequence of preserving adjacency is that, for all one dimensional functions, Gray codes will not create any additional local optima when compared to the original real-valued function [109]. Because Standard Binary destroys the natural adjacency, it can create local optima that do not exist in the original function. This argument supports empirical evidence that Standard Binary Reflected Gray code is usually a more effective representation than Standard Binary [109]. For this reason, the discrete representation used throughout this thesis is Standard Binary Reflected Gray code.

2.2 Genetic Algorithms

Genetic algorithms traditionally represent individuals in the search space as finite bit strings. Through a process of *selection* and *reproduction* the current population of individuals is transformed into a new generation. Selection dictates which individuals in the current population will have a chance to reproduce. The goal of selection is to allocate more reproductive opportunities to above average individuals in the population. This creates a bias called *selective pressure*. Reproduction usually involves both crossover and mutation. In most genetic algorithms, the *crossover rate* is set much higher than the *mutation rate*, making the *crossover* operator responsible for more of the variation that occurs between generations. The crossover operator combines two selected parents by concatenating specific pieces of each parent's string together. Mutation randomly changes part of an individual as a means of exploration and diversification.

2.2.1 CHC

One of the more effective variants of the traditional genetic algorithm is CHC [32]. CHC is different because it selects two parents for recombination in a uniform random way —there is no bias toward selecting better individuals. Instead, selective pressure is created using *cross-generational* truncation selection; newly created offspring must compete with the parent population for survival.

CHC also uses a modified version of uniform crossover, where half of the non-matching bits are exchanged. The children under this scheme are always the same maximal Hamming distance from both parents. Further steps are taken to ensure that parents are not allowed to mate unless they are sufficiently different. Eshelman refers to this as *incest prevention* [32]. Initially, two strings must be different by at least $L/4$ bits, where L is the length of an individuals string. This *difference threshold* decreases by 1 each time crossover fails to produce an improving individual. This means that over time, crossover may occur on strings that are more similar. Eshelman states that incest prevention coupled with a difference threshold enables CHC to perform a coarse grain search initially, which preserves diversity, and eventually resort to a more fine grain exploitive search.

No mutation is used to alter one generation to the next. Instead, when the difference threshold decreases to zero, CHC initiates a restart mechanism called *cataclysmic mutation*, that reinitializes the entire population by randomly flipping 35% of the bits of the best individual. The entire CHC process is shown in Figure 2.1.

Table 2.1 shows the parameters specific to CHC. One advantage in using CHC is that almost no parameter tuning is needed. The population size can be changed, but Whitley *et al.* have shown that increasing the population size from the standard value of 50 usually results in a performance loss [103]. Precision is the only other consideration. Typically, this is set between 10- and 20-bits. A higher precision search will tend to restart less often because each individual's representation considers more information, and therefore, will maintain diversity longer.

Symbol	Parameter	Range	Default
λ	Population size	$\mathbb{N}$	50
b	Precision	$\mathbb{N}$	10

Table 2.1: The default parameter settings for CHC.

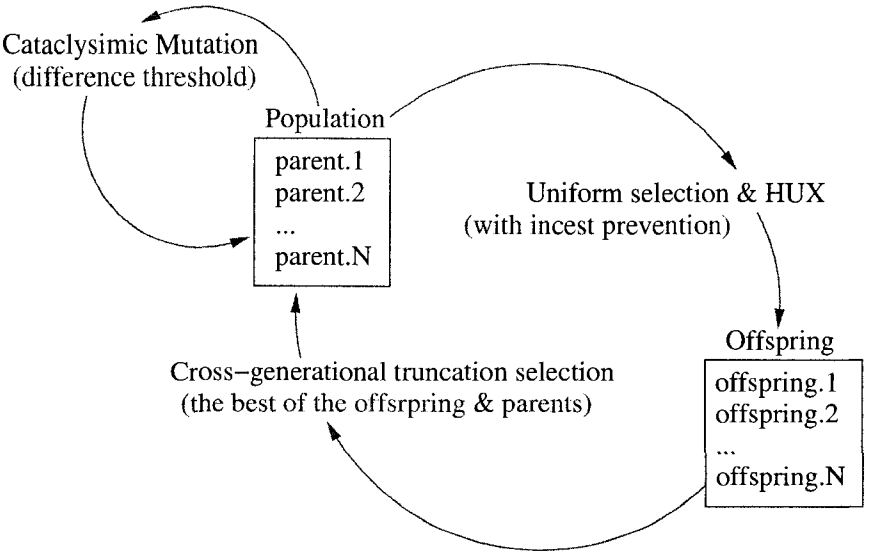


Figure 2.1: The CHC genetic algorithm.

2.3 Evolution Strategies

The canonical *evolution strategy* is an iterative process where a population of μ parents produce λ offspring based on mutation distributions that center around the parents. Evolution strategies differ from genetic algorithms in that individuals in the population are generally represented as real-valued vectors instead of discrete bit-strings. Evolution strategies also use a deterministic selection scheme instead of a stochastic one. In a (μ, λ) selection strategy, the new population is created by choosing the best μ individuals from the λ offspring. An elitist strategy, on the other hand, selects the μ best individuals from a set that contains both the parents and the offspring. This is known as a $(\mu + \lambda)$ selection strategy.

Since evolution strategies use a real-valued representation, *discrete recombination* is not used. Instead, a form of *continuous recombination* creates a single offspring based on a linear combination of ρ parents. The *mixing number*, ρ, refers to the number of parents that are combined together to produce a single offspring. When $\rho = 1$, no recombination occurs. *Intermediate recombination* is a special case where $\rho = \mu$. Here, a single offspring is created based on the average position of the current best μ individuals. Sometimes a weighted average is used in recombining the parents. The notation of the canonical evolution strategy using recombination is $(\mu/\rho, \lambda)-$ES.

Mutation in an evolution strategy is defined by a probabilistic distribution. The way in which

these mutation distributions are defined distinguishes one evolution strategy from another. In this paper, we consider two ways of adapting mutation distributions: *self-adaptation* and *covariance matrix adaptation.*

2.3.1 Self-Adaptation in Evolution Strategies

In self-adaptation, each individual in the population is described by a set of *object parameters*, which define its location in the search space, and a set of *strategy parameters*, that define its mutation distribution. An individual's fitness is derived directly from its location in the search space (object parameters). In general, individuals with higher fitness, and therefore better object parameters, are more likely to survive. Unlike object parameters, strategy parameters are not directly optimized. One of the defining assumptions made by self-adaptation is that the best individuals of the current generation are likely to have strategy parameters that will create better children in the coming generations.

Bäck describes the typical evolution strategy for self-adapting a global step-size [8]. In the following equations, $N(0,1)$ denotes a normally distributed random number with mean 0 and a standard deviation of 1. Before the *object parameters* are created, the step-size for each offspring is adapted.

$$\sigma^{g+1} = \sigma^{g} \cdot \exp(N(0,\tau^{'}))$$

Here $\tau^{'}$ is the *global mutation strength* on σ. This determines how quickly σ can change from one generation to the next. We use the standard value of $\tau^{'} = 1/\sqrt{2n}$. Once each individual has been assigned a new value for σ, its object parameters are created based on this new distribution.

$$x_i^{g+1} = x_i^{g} + \sigma^{g+1} \cdot N_i(0,1)$$

Note that, for each offspring, a new random number is sampled for each dimension, denoted $N_i(0,1)$, scaled by σ^{g+1}, and translated around the parent x_i^g.

The specific parameters used by self-adaptation are shown in Table 2.2. Bäck and Schwefel assert that the ideal ratio of μ/λ is $1/7$ [10]. The initial step-size is problem dependent. Beyer and Schwefel suggest decreasing the learning rate $\tau^{'}$ for multimodal surfaces.

Symbol	Parameter	Range	Default
μ	Number of parents	$\mathbb{N}$	1
λ	Population size (# of offspring)	$\mathbb{N}$	$7 \cdot \mu$
ρ	Mixing number	$\mathbb{N}$	1
σ_0	Initial step-size	$\mathbb{R}^+$	1
τ'	Learning rate	$\mathbb{R}^+$	$1/\sqrt{2N}$

Table 2.2: The default parameter settings for *self-adaptation* of a single global step-size. The value of N represents the problem dimension.

Self-adaptation can be extended to adapt elliptical distributions defined by individual step-sizes, but this mutation distribution has a tendency to collapse and search only a subset of the search space [15; 72; 43; 70; 47; 62]. Figure 2.2 graphically illustrates this problem. The elliptical distributions represent the step-sizes for two parameters of the best individual from each generation. For clarity, not all generations are shown. As search progresses, the step size in one parameter starts to shrink. This creates a long elliptical distribution that has the potential to highly influence future generations. Without a lower bound on the step-size, the distribution eventually decrease to machine precision, causing search to stagnate in that dimension.

Correlated mutations extends self-adaptation even further by estimating the covariance for each pair of object parameters, which, in theory, can describe any elliptical distribution in the search space. But Hansen *et al.* show that using correlated mutations becomes impractical when the dimensionality is higher than about ten [43] and that the performance of correlated mutations is strongly related to the initial values of the strategy parameters [39]. These shortcomings suggest that self-adapting more than one strategy parameter is unreliable.

2.3.2 Covariance Matrix Adaptation

Hansen and Ostermeier introduced *Covariance Matrix Adaptation*, or CMA, as an alternative to *correlated mutations* that uses a covariance matrix to explicitly rotate and scale the mutation distribution [43]. The orientation and shape of the distribution is not indirectly adapted, as in self-adaptation, but calculated based on the *search path* and, when applicable, *local information* extracted from large populations. This makes CMA-ES extremely efficient on many poorly scaled and non-separable functions. Hansen and Ostermeier define the reproduction phase from generation g to generation

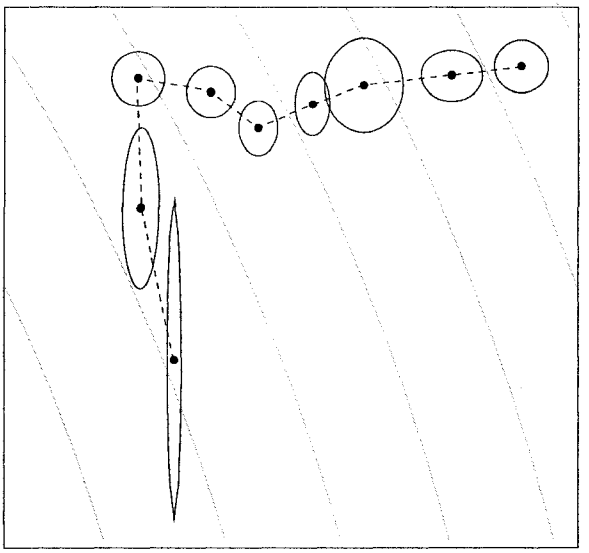


Figure 2.2: The above figure is a two dimensional slice of the path taken by an evolution strategy while operating with individual step-sizes. The x-axis is parameter 21 and the y-axis is parameter 22 from a single trial of a simple 30 dimensional function. The black circles show the mutation distribution defined by the individual step-sizes associated with parameters 21 and 22 of the best individual of the generation (black dot). Without a lower bound on the step-size, the value of the object parameter associated with dimension 21 will not be able to move, causing search to stagnate.

$g+1$ as:

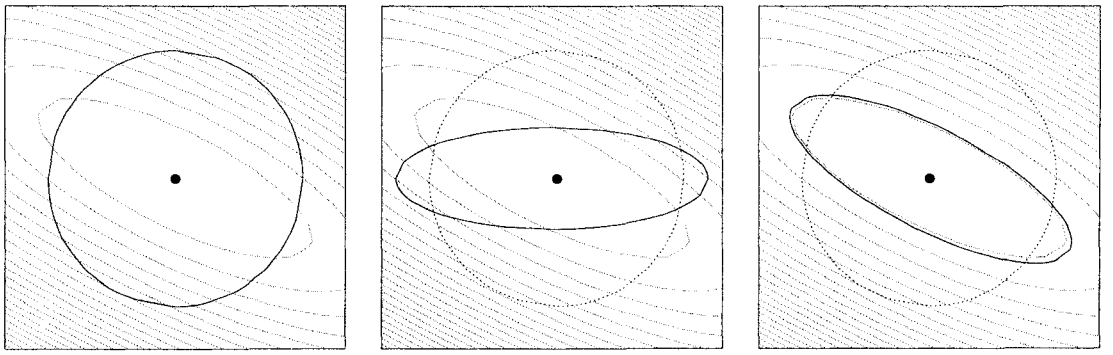
$$x_k^{(g+1)} = \langle x \rangle_\mu^{(g)} + \sigma^{(g)} \mathbf{B}^{(g)} \mathbf{D}^{(g)} z_k^{(g+1)}$$

where $z_k^{(g+1)}$ are randomly generated from an $N(0, I)$ distribution. This creates a set of base points that are rotated and scaled by the eigenvectors ($\mathbf{B}^{(g)}$) and the square root of the eigenvalues ($\mathbf{D}^{(g)}$) of the covariance matrix $\mathbf{C}$. The single global step size, $\sigma^{(g)}$ is used to scale the distribution. Finally, the points are translated to center around $\langle x \rangle_\mu^{(g)}$, the mean of the μ best parents of the population.

The objective of CMA-ES is to adapt a covariance matrix that fits the contour lines of the objective function (Figure 2.3). This goal is partly achieved by using a time dependent portion of the *evolution path* to adapt the covariance matrix. The evolution path is a vector that points in the *recent* overall direction search has progressed. The evolution path is calculated as

$$\vec{p}_c^{(g+1)} = (1-c) \cdot \vec{p}_c^{(g)} + \sqrt{c(2-c)} \cdot \frac{\sqrt{\mu_{\text{eff}}}}{\sigma^g} \left(\langle \vec{x} \rangle^{(g+1)} - \langle \vec{x} \rangle^{(g)} \right)$$

where $\langle \vec{x} \rangle^{(g+1)} - \langle \vec{x} \rangle^{(g)}$ is the vector representing the current step. Intermediate recombination is used to create $\langle \vec{x} \rangle^{(g)}$, which represents the mean of the μ best points of generation g. The relative



$\sigma N(0, I)$ $\sigma \mathbf{D} N(0, I)$ $\sigma \mathbf{B}\mathbf{D} N(0, I)$

Figure 2.3: The CMA evolution strategy: The leftmost figure represents a distribution defined by a single strategy parameter, σ. The middle figure requires n strategy parameters and yields a distribution that is an axis-aligned ellipsoid. The far right distribution uses a full covariance matrix and is capable of fitting the contours of a quadratic objective function.

importance of previous steps is weighted by c, the cumulative time parameter. When $c = 1$, no history is considered. The value for c is usually on the order of $1/n$. The value $\sqrt{\mu_{\text{eff}}}/\sigma^g$ is a normalization constant [43]. The covariance of the evolution path is the rank-one update of the covariance matrix $\mathbf{C}$.

When a larger population is used, the steps of the best μ individuals may help describe the topology around the mean of the current generation [42]. This may increase the accuracy of the covariance matrix estimation. In order to exploit this information, CMA-ES uses a rank-μ-update, that calculates the covariance of the steps that lead to the best μ individuals:

$$\mathbf{Z}^{(g+1)} = \frac{1}{\mu} \sum \mathbf{B}^{(g)} \mathbf{D}^{(g)} z_i^{(g+1)} \left(\mathbf{B}^{(g)} \mathbf{D}^{(g)} z_i^{(g+1)} \right)^T$$

This information, along with the evolution path, $p_c^{(g)}$, is used to update the covariance matrix. Assuming $\mathbf{Z}^{(g+1)}$ is the covariance of the steps leading to the best μ individuals, and $\mathbf{P}^{(g+1)}$ is the covariance of the evolution path, the new covariance matrix is

$$\mathbf{C}^{(g+1)} = (1 - c_{cv}) \mathbf{C}^{(g)} + c_{cv} \left(\alpha_{cv} \mathbf{P}^{(g+1)} + (1 - \alpha_{cv}) \mathbf{Z}^{(g+1)} \right),$$

where c_{cv} and α_{cv} are constants that weight the importance of each input. A principal components analysis on the covariance matrix extracts the eigenvectors $\mathbf{B}^{(g)}$ and the square root of the eigenvalues $\mathbf{D}^{(g)}$. This information allows CMA-ES to construct elliptical mutation distributions that are unbiased to any orientation in the search space. Figure 2.4 illustrates this idea in two dimensions.

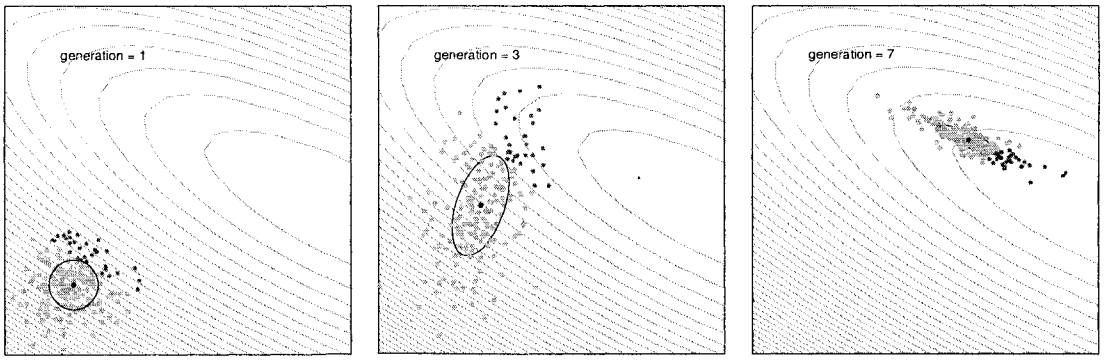


Figure 2.4: The rank-μ update: The light gray points are sampled from the distribution defined by the black ellipse. The black points are the best of the sample. Using the steps to these points to update the covariance matrix is called the rank-μ update.

The global step-size, $\sigma^{(g)}$, is adapted using *cumulative step-size adaptation* (CSA). Introduced by Ostermeier et al. [73], CSA uses the *evolution path* (not scaled by the eigenvalues $\mathbf{D}^{(g)}$ of $\mathbf{C}^{(g)}$) to determine parallel, or anti-parallel steps. When the evolution path is longer than expected, the search steps are probably parallel and the mutation strength should increase and decrease otherwise. Figure 2.5 illustrates this heuristic.

The expected length of the evolution path under *random selection* is simply the expected length of a random normal vector ($E||N(0,I)||$). This is approximated as:

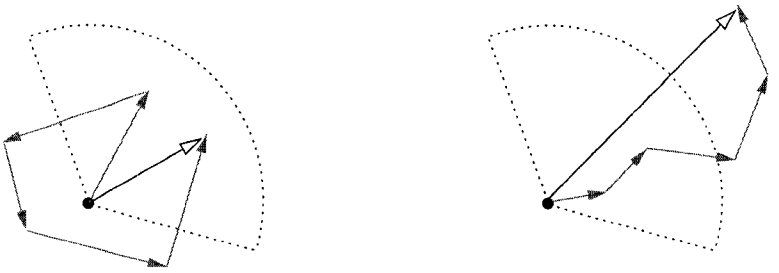
$$E||N(0,I)|| = \chi_n = \sqrt{n}\left(1 - \frac{1}{4n} + 1 - \frac{1}{21n^2}\right)$$

The strategy parameter defining the global step size is updated as follows:

$$\sigma^{(g+1)} = \sigma^{(g)} \cdot \exp\frac{c}{d}\left(\frac{||\vec{p}_\sigma^{\,g+1}|| - 1}{\chi_n}\right)$$

Here, d is the damping factor, whose default is 1. The evolution path $\vec{p}_\sigma^{\,g+1}$ is calculated such that it is not scaled by $\mathbf{D}^{(g)}$ (e.g. not to be confused with $\vec{p}_c^{\,g+1}$).

The default parameter settings for CMA-ES are listed in Table 2.3. Increasing the population size will increase the learning rate of the global step-size $\sigma^{(g)}$, but has also been shown to increase the robustness of CSA (CMA without matrix adaptation) in noisy environments [4]. Increasing the population size has also been shown to increase the performance of CMA on multimodal functions [41; 7]. Decreasing the cumulative time parameter will result in a evolution path that includes a greater number of generation steps. The initial step-size is problem dependent. If σ_0 is too small,



Decrease σ Increase σ

Figure 2.5: Step-size control using cumulation. The gray arrows indicate generational steps. The dashed line is the expected length of a random normal vector ($E||N(0, I)||$) and the black line with the white arrow is the length of the evolution path $\vec{p}_c$. When the evolution path is shorter than expected, σ should decrease. When the path is longer than expected, the generational steps are probably parallel and the step-size should increase.

the performance of CMA will be dependent on where in the search space it is initialized. By default, CMA uses *weighted recombination*. Intermediate recombination can also be used where $w_i = 1$ for all $i \in 1 : N$.

Symbol	Parameter	Range	Default
μ	Parent number	$\mu < \lambda$	$\mu = \lfloor \lambda/2 \rfloor$
λ	Population size (# of offspring)	$\mathbb{N}$	$4 + \lfloor 3 \ln N \rfloor$
σ_0	Initial step-size	$\mathbb{R}^+$	1
c	Cumulative time	$0 < c < 1$	$1/\sqrt{N}$
w_i	Weight of the i^{th} parameter	$\mathbb{R}^+$	$\ln \frac{\lambda+1}{2} - \ln i$

Table 2.3: The default parameter settings for CMA. In the above equations, N is the problem dimension.

2.4 Pattern Search

Pattern search methods differ from evolutionary algorithms in that they are primarily deterministic. The underlying performance of a typical pattern search algorithm resembles that of basic local search. At every iteration k, the current solution x_k is compared to a finite set of points either on a next-descent or best-descent basis. The deterministic pattern of finite points is used to create variation on the current solution, x_k. This takes the place of stochastic operators, such as *recombination*

and *mutation*, that are used in evolutionary algorithms. Selection is also deterministic; the current solution for the $k+1$ iteration is chosen from the pattern of finite points and the best solution from iteration k.

The simplex method of Nelder and Mead is probably the most widely used direct search algorithm. Surprisingly, this method is known to fail on relatively simple problems. We describe this method for historic reasons and then discuss more advanced pattern search methods, such as *Generalized Pattern Search* (GPS) [96] and *Mesh Adaptive Direct Search* (MADS) [6].

2.4.1 Nelder and Mead Simplex

Spendly, Hext, and Himsworth [92], noticed that, given an optimization problem with n parameters, it only requires $n+1$ calls to the objective function to identify the estimated gradient direction of the surface. This is the basic idea of their original *simplex* search.

Once the initial simplex is formed, each vertex is evaluated. Of the $n+1$ points evaluated, the vertex with the least desirable fitness is singled out as the candidate for replacement, and is replaced by reflecting the point through the centroid of the remaining vertexes. These reflection steps allow the simplex to "flip" down the surface in the direction of the gradient.

Nelder and Mead applied the simplex algorithm of Spendly, Hext, and Himsworth [92] to optimize nonlinear functions. They added additional expansion and contraction moves designed to increase the efficiency of the search [69]. For example, if the reflection move provided a new best solution, they reasoned that was advantageous to continue searching in that direction. If neither the reflective step, nor the expansion step improve the current best solution, Nelder and Mead assume that the simplex is straddling the minimum, and a contraction step is necessary. The contraction moves from the centroid, either toward the reflection step or away. If neither of the steps prevail as a new best solution, the Nelder-Mead simplex resorts to shrinking the length of the simplex, similar to the algorithm defined by Spendly, Hext, and Himsworth. Figure 2.6 graphically shows how the Nelder-Mead simplex expands and contracts.

2.4.2 Generalized Pattern Search

Generalized Pattern Search (GPS) refers to a group of deterministic search algorithms that are unified under an abstract definition developed by Torczon [96], that includes early pattern search meth-

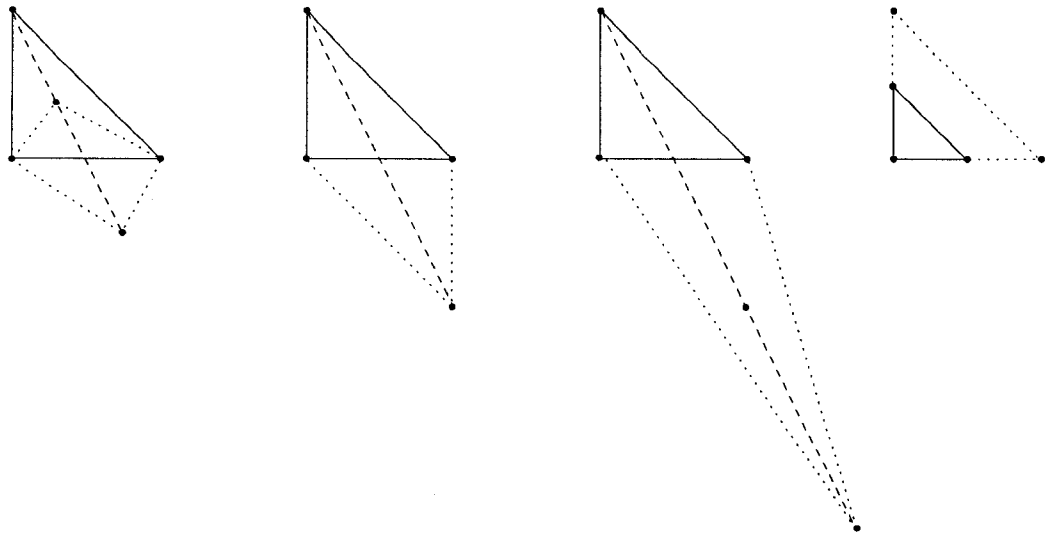


Figure 2.6: The simplex method of Nelder and Mead. The first graph on the left shows the two contraction steps that move either toward the reflection step or away. If the reflection step provides improving fitness, then the expansion step searches further in that direction. Finally, if all steps fail, the simplex shrinks.

ods such as *Evolutonary Operation* proposed by Box [18] and Hooke and Jeeves' pattern search [46], as well as her own *multi-directional search* [95]. If constructed appropriately, generalized pattern search methods are guarantee to converge to a local optima within a predetermined degree of precision.

At each iteration k, GPS methods create variation by evaluating points that belong to a mesh grid constructed from a matrix of directions D and a step size σ. Together, D and σ create a neighborhood of potential candidate solutions. The search directions must remain constant during the course of search, which means that arbitrary search directions are not allowed once D has been initialized. Each *pattern* must contain at least $n + 1$ points in order to guarantee a descent direction, where n is the number of dimensions.

Although the *shape* of the pattern cannot change, the *size* of the initial pattern can. Because of this restriction, the progress of these methods largely depends on the choice of the initial search directions [95]. The step-size update rule is simple and intuitive: if none of the neighborhood points (σ D) offer an improvement, the step-size is decreased by half, and doubled otherwise. Torczon and Trosset point out that pattern search methods pay a price for a convergence guarantee because they may need to evaluate as many as $n + 1$ points in order to alter the step-size [97]. This is especially

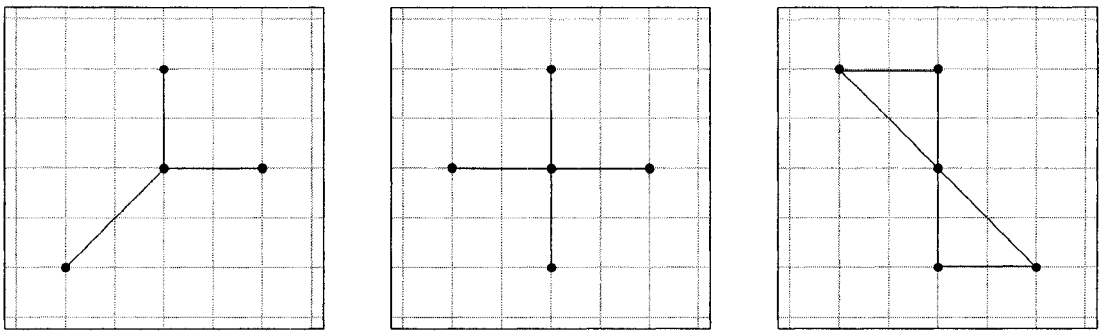


Figure 2.7: A selection of patters allowed by GPS. Notice that each pattern maps onto a lattice.

problematic in higher dimensions.

Any pattern with at least $n + 1$ points is admissible as long as they map onto a rational lattice. Figure 2.7 shows three possible patterns. The leftmost pattern is a simplex. The first n directions are the positive unit vectors along each dimension axis. The $(n + 1)^{st}$ direction is taken to be the negative sum of the first n dimensions. We label this implementation GPS(n+1). The middle pattern evaluates points on the positive and negative coordinate axis, and is denoted GPS(2n). The pattern on the right is Torczon's multi-directional search.

2.4.3 Mesh Adaptive Direct Search

Audet and Dennis [6] introduce *Mesh Adaptive Direct Search* (MADS) in order to add flexibility to the GPS framework. The main difference between MADS and GPS is that the neighborhood pattern, defined by D, is not limited to a finite number of directions once initialized. The search directions can be chosen arbitrarily, and randomly, as long as they span the search space.

As with GPS, the points are taken such that they belong to a mesh constructed from a matrix of directions and a step size. However, additional flexibility allows the directions of the pattern to change by specifying a *mesh size* parameter, Δ. The directions are still specified by the columns of D and, as mentioned, can be chosen arbitrarily with the only requirement that D spans R^n. Δ can be thought of as the granularity of the mesh. Its value is increased if an improving point is located, and decreased otherwise.

We used the NOMAD package [21] as a code base for running experiments with GPS and MADS. In each instance, the step size, σ, was adapted as already described for GPS. This software

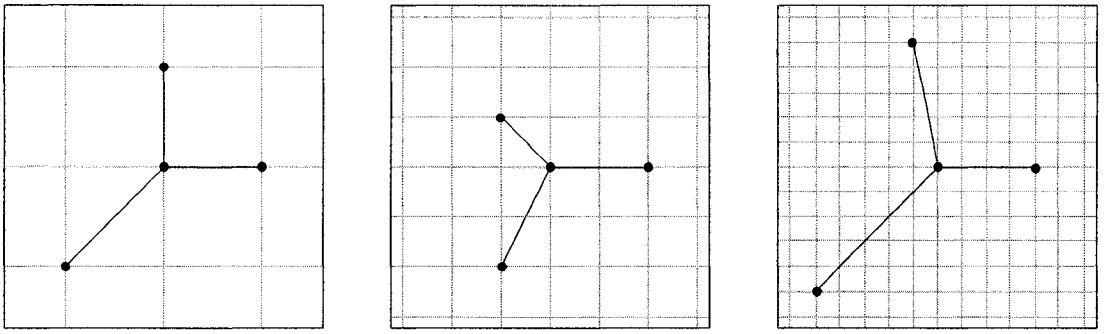


Figure 2.8: Mesh Adaptive Direct Search: Adapting the granularity of the mesh allows MADS to search in arbitrary directions as long as the random pattern still spans the search space. In the above graphics, the mesh size is $\Delta = 1$, $\frac{1}{2}$, and $\frac{1}{4}$.

contains an implementation of five direction sets that, for this document, we label as: GPS(2n), GPS(n+1), Unif(n+1), MADS(2n), and MADS(n+1). The set Unif(n+1) is a simplex and its directions radiate from the current iterate uniformly through the search space (e.g. vertices of an equilateral triangle, regular tetrahedron, etc.), rather than being aligned with dimension axes.

The initial pattern directions of MADS(2n) and MADS(n+1) are the same as those previously described for GPS(2n) and GPS(n+1). The sets MADS(2n) and MADS(n+1) should really be thought of as higher-resolution versions of their GPS counterparts. That is, MADS is not limited to step-size variations of the initial pattern, but can adapt arbitrary directions. For example, because the directions in GPS(2n) are aligned with the search space axes, the search is expected to creep along any encountered ridges. MADS allows the search to progress along directions that are not necessarily aligned with the axes. The number of such possible directions is determined solely by the ratio σ/Δ. Figure 2.8 shows how MADS can allow for arbitrary search directions using the mesh grid parameter Δ. For MADS, Δ was multiplied by four at each iteration if an improving point is found and divided by four otherwise.

Typically, a pattern search algorithm is terminated when the step size falls below a certain threshold. For comparison reasons, we are interested in using the number of function evaluations as terminating conditions. Thus a restart mechanism was used when the step size became too small: search continues to restarted from a random position until some maximum number of function evaluations.

Symbol	Parameter	Range	Default
b	Precision	$\mathbb{N}$	10

Table 2.4: The default parameter settings for local search.

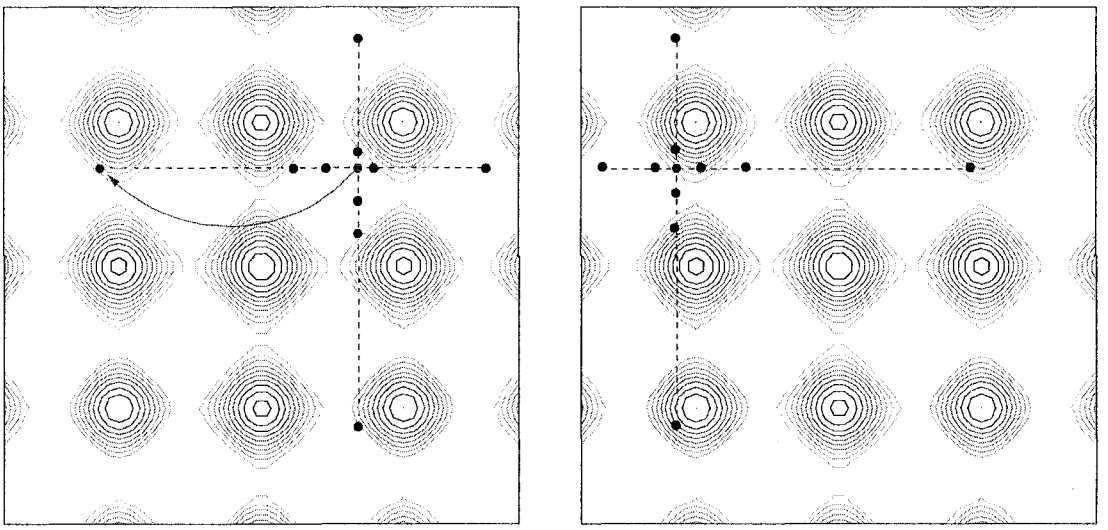


Figure 2.9: Local Search: After evaluating each point in the coordinate pattern, local search moves to the best point (this move is shown with an arrow). The process repeats until no improvements are found.

2.4.4 Local Search

In this paper, local search refers to a *steepest ascent bit climber*, where each parameter is encoded as a bit-string using the same technique employed by the CHC discrete genetic algorithms. A neighborhood pattern forms around the current best solution by flipping one bit at a time. Local search evaluates all these neighborhood points before taking the best, or *steepest*, step. Local search restarts when no improving move is found. Figure 2.9 illustrates this idea.

The only parameter for local search is the precision, which is expressed as the number of bits used to represent each parameter. This makes local search extremely easy to use. The default values are typically set between 10- and 20-bits. A higher precision will converge more slowly because the neighborhood pattern is larger. This becomes a practical problem in high dimension. For example, on a 100 dimensional problem with 20-bits of precision, the neighborhood size of local search is $100 \cdot 20 = 2000$. The can be cut in half using 10-bits of precision, but it will still take 1000 evaluations just to take a single step.

2.5 Gradient-based Methods

The methods described so far search the objective function using only the values it returns (e.g. direct-search). Gradient-based methods require the objective function to be differentiable. Unlike evolutionary algorithms, gradient-based methods are strictly local search methods, meaning they will only find the local optima that exists in the basin of attraction for which they are initialized.

The gradient of a function at a particular point is a vector pointing in the direction that represents the greatest rate of increase. The "steepness" of the function is given by the magnitude of the gradient. This is often interpreted as the fall line of a hill. Mathematically, if

$$f(x) = f(x_1, x_2, ...x_n)$$

is the objective function we are optimizing, then the gradient is the vector of first derivatives:

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, ..., \frac{\partial f(x)}{\partial x_n}\right]^T$$

The Hessian of a function is less intuitive than the gradient. Essentially, the Hessian at a particular point measures the degree to which the parameters of the objective function function interact. The Hessian is the matrix of second derivatives:

$$\nabla^2 f(x)_{i,j} = \frac{\partial f(x)}{\partial x_i \partial x_j}$$

2.5.1 Steepest-descent

The method of *steepest-descent* iteratively line searches in the direction of the gradient to minimize a function. Formally, if our current point is x_k, then steepest-descent finds the step-size, α, that minimizes the objective function in the direction of the negative gradient, $-\nabla f(x)$. An iteration moves search from x_k to x_{k+1} as:

$$x_{k+1} = x_k - \alpha \nabla f(x)$$

A line search method finds the step-size α that minimizes

$$f(x_k - \alpha \nabla f(x))$$

Figure 2.10 graphically shows this process. The leftmost figure shows the direction of the gradient from the point x_k as a dashed line. The one-dimensional curve in the right most figure

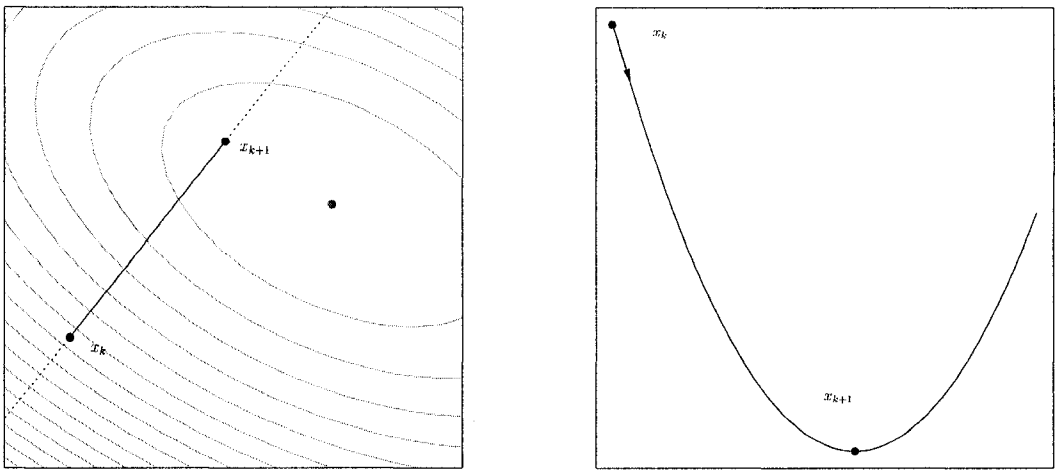


Figure 2.10: The method of steepest-descent: A line search in the direction of the gradient (dashed line) leads to the minimum in that direction, x_{k+1}.

shows a slice of the objective function along this line, which is the direction of the gradient. A line search procedure will find the value of α such that $x_k - \alpha \nabla f(x)$ is minimized. This results in the new point x_{k+1}. The length of the black line in the leftmost figure represents the value of step-size α.

As it turns out, simply following the gradient is an inefficient way of minimizing most functions. In fact, the efficiency of steepest-descent is directly proportional to how poorly scaled the objective function is. We discuss this in greater detail in chapter 4, but point out here that, for this reason, steepest-descent can be inefficient when used as a gradient-based local optimizer. The conjugate-gradient method, and the class of quasi-Newton methods, use the gradient in a much more efficient way.

2.5.2 Quasi-Newton Methods

All quasi-Newton methods are based on Newton's method. Newton's method starts with the first-order necessary condition for a local optima, which states that the gradient of a particular point be equal to zero.

$$\nabla f(x) \quad = \quad 0$$

Then Newton's method makes a quadratic approximation of the objective function, $f(x)$, by using the first three terms of the Taylor series expansion for f at the point x_k:

$$q(x) = f(x^k) + (x - x^k)^T \nabla f(x^k) + \frac{1}{2}(x - x^k)^T \nabla^2 f(x^k)(x - x^k)$$

This creates a quadratic surface, $q(x)$, that approximates the nonlinear function $f(x)$. Then Newton's method finds the minimum of $q(x)$ by setting its gradient equal to zero. This constitutes an iteration of Newton's method. The standard form is best understood through the following derivation. First, we minimize the approximate function $q(x)$.

$$\nabla q(x) = \nabla f(x^k) + \nabla^2 f(x^k)(x - x^k) = 0$$

Solving this equation for x gives:

$$\begin{aligned} \nabla f(x^k) + \nabla^2 f(x^k)(x - x^k) &= 0 \\ \nabla^2 f(x^k)(x - x^k) &= -\nabla f(x^k) \\ x - x^k &= -[\nabla^2 f(x^k)]^{-1} \nabla f(x^k) \\ x &= x^k - [\nabla^2 f(x^k)]^{-1} \nabla f(x^k) \end{aligned}$$

The point x represents the minimum of the quadratic approximation. In practice, the inverse Hessian $[\nabla^2 f(x^k)]^{-1}$ is not computed. Instead it is more efficient to solve the system of equations for the vector $p = x - x_k$.

$$\nabla^2 f(x_k) \;\; p = \nabla f(x_k)$$

This gives the *direction* and *step-length* that minimizes $q(x)$. In other words, given p, the next step is:

$$x_{k+1} = x_k + p$$

The effectiveness of this step is proportional to how well the quadratic surface $q(x)$ approximates the objective surface $f(x)$. On quadratic functions, it will find the exact solution in one step. On some nonlinear functions, it may only requires a few steps. But this method can fail. If the Hessian $\nabla^2 f(x^k)$ is not positive definite, then the search direction is no longer a descent direction, and

the next iteration may return a value that is greater than the current iterate (e.g. $f(x_{k+1}) > f(x_k)$ may occur). Newton's method is also costly; computing the Hessian requires second derivatives, which are often complex and expensive.

Quasi-Newton methods increase reliability and decrease complexity by approximating the Hessian $\nabla^2 f(x^k)$ in a lower cost way that ensures the resulting matrix is still positive definite. This is a trade-off between algorithm complexity and number of iterations. Quasi-Newton methods require more iterations, but do not require second derivatives. Instead, the Hessian $\nabla^2 f(x^k)$ is approximated by a matrix B_k using only the gradient. Then a step-size is computed by solving the following system of equations:

$$B_k \quad p = -\nabla f(x_k)$$

Since B_k is only an approximation, a line search finds α such that an optimal step is taken in the direction of p:

$$x_{k+1} = x_k + \alpha p$$

In Newton's method, $\alpha = 1$. Before the next iteration, the approximate Hessian B_k is updated based on information from this step. The way that this update occurs often distinguishes one quasi-Newton method from the next.

Steepest-descent can be thought of as a quasi-Newton method that uses the identity matrix as an approximation to the Hessian. On functions that are not poorly scaled and have limited parameter interaction, this is a reasonable approximation and steepest-descent works well. When this is not the case, the direction used by steepest-descent can be an inefficient choice. Figure 2.11 graphically explains this difference.

In chapters 5 and 7 we compare evolutionary search with the BFGS [20; 34; 37; 89] quasi-Newton method, which is named after its creators: Broyden, Fletcher, Goldfarb, and Shanno. The details of this update can be found in any introductory optimization text, such as Nash and Sofer [68] "Linear and Nonlinear Programming", and are not critical for the work presented here.

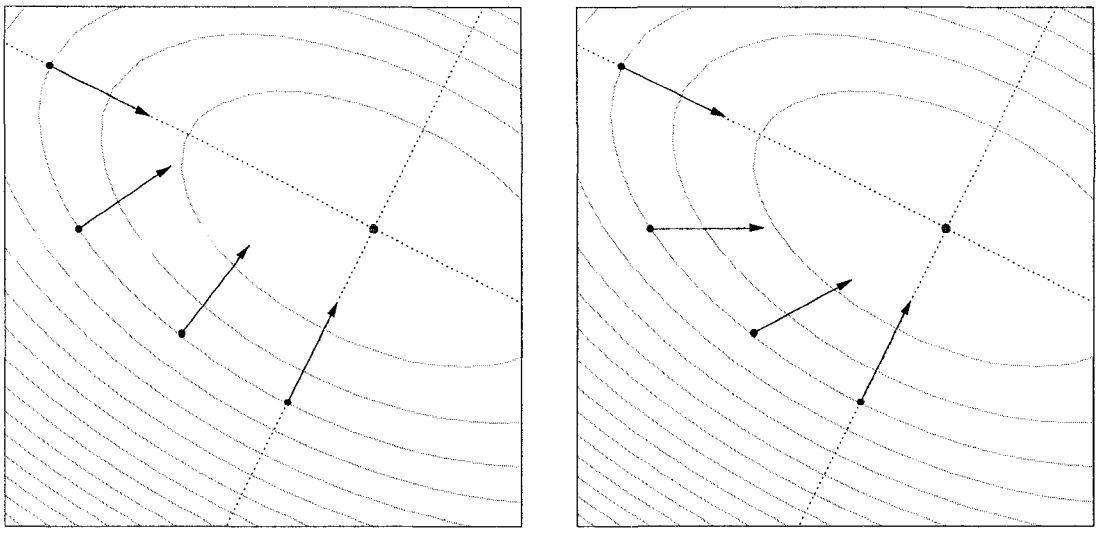


Figure 2.11: Steepest-descent and Newton's method: The figure on the left is steepest-descent, which can be thought of as a quasi-Newton method where the identity matrix approximates the Hessian. In this instance, the search directions point in the direction of the greatest rate of change, which is rarely toward the optimum. Knowing the Hessian allows these directions to correct for the scale and interaction of the parameters. This means that the search directions used in Newton's method will point in the direction of the optimum of a quadratic function (right figure).

2.5.3 Conjugate Gradient Method

Conjugate gradient methods [35] exploit the fact that *steepest descent* often takes steps in the same direction, multiple times. Assuming an algorithm could find the right set of directions, convergence could be increased by taking fewer steps in the correct directions. The goal, therefore, of the *conjugate gradient method* is to find a good approximation to this optimal set of directions.

The algorithm starts at a point x_0 and takes the steepest step.

$$\begin{aligned} d_0 &= -\nabla f(x) \qquad \text{(gradient direction)} \\ x_1 &= x_0 + -\alpha\, d_0 \end{aligned}$$

Instead of repeating the next iteration in the steepest direction, the conjugate gradient method computes a search direction that is "conjugate" to the previous direction as opposed to "orthogonal", as is the case with steepest descent. This new direction is calculated by first computing an adjustment factor c_k.

$$c_k = \frac{(\nabla f(x_k) - \nabla f(x_{k-1}))^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}$$

This particular update of c_k is known as the *Polak-Ribiere* formula. Then, instead of searching in the direction of the gradient ($-\nabla f(x_k)$) at iteration k, the new direction is adjusted based on c_k and the previous search direction d_{k-1}.

$$d_k = -\nabla f(x_k) + c_k d_{k-1}$$

The conjugate gradient method does not store an approximation of the Hessian matrix. This makes it suitable for larger dimensional problems where storage is a concern. This method is also very efficient because it does not require solving a system of equations that is necessary in quasi-Newton methods.

The direct search community has recognized the merits of this algorithm. In his paper titled, "An efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives", Powell outlined a way to approximate this method by generating one-dimensional line searches [79]. But this method can fail. Zangwill demonstrated that if no success is obtained in one of the search directions, the directional vectors would no longer span the search space [112]. Brent outlines several modifications to Powell's method, which he called PRAXIS, or principal axis [19], that solves the problem of linear dependency among search vectors by resetting the search directions after each iteration. Instead of resetting the search directions back to the coordinate axis, Brent proposes finding principal vectors that are computed under the assumption that the objective function is quadratic.

2.5.4 Levenberg-Marquardt

There are several objective functions that have the following basic form:

$$\begin{aligned} f(\vec{x}) &= \frac{1}{2}\sum_{i=1}^{M} r_i(\vec{x})^2 \\ &= \frac{1}{2} r(\vec{x})^T r(\vec{x}) \end{aligned}$$

The objective is to find the vector $\vec{x}$ that minimizes $f(\vec{x})$, meaning that each of the m sub functions r_i are indirectly minimized also. Often these sub functions are thought of as residuals and are expected to go toward zero when $f(\vec{x})$ is minimized. Objective functions of this nature are called nonlinear least-squares problems.

Any of the algorithms we have discussed so far can be used to minimize an objective function of this form. However, because of the structure of the objective function, the Hessian matrix has a distinctive form; it is the sum of two terms: the first only involves the gradient and the second is zero when the residuals r_i are zero. Algorithms that are specifically designed to solve nonlinear least-squares problems exploit this unique structure.

Motivation: Gauss-Newton method

If we applied Newton's method to a nonlinear least-squares problem, we would solve the following system of equations to find the vector p that minimized the underlying quadratic approximation function based on a Taylor series expansion of $f(x)$.

$$\begin{aligned} \nabla^2 f(x)\ p &= \nabla f(x) \\ \left(J(x)^T J(x) + S(x)\right)\ p &= J(x)^T r(x) \end{aligned}$$

Here $J(x)$ is the Jacobian of $f(x)$ at the point x. The term $S(x)$ is defined as:

$$S(x)_{i,j} = r_i(x)\frac{\partial^2 r_i(x)}{\partial x_i \partial x_j}$$

Instead of using the actual Hessian, the Gauss-Newton method approximates the Hessian using only first-order information.

$$B_k = J(x)^T J(x) \approx \nabla^2 f(x_k).$$

The rational behind this approximation is simply that when $r_i(x) \approx 0$, then $S(x) \approx 0$ and the true Hessian $\nabla^2 f(x_k) \approx J(x)^T J(x)$.

The best of both worlds

Although there are several problems with this approach, the primary complaint is that when the residual values $r_i(x)$ are not small, $J(x)^T J(x)$ is a poor approximation of the true Hessian. It is also possible that $J(x)^T J(x)$ becomes nearly singular and may not be positive definite.

The Levenberg-Marquardt [53; 61] algorithm adds a positive parameter λ to the approximation of the Hessian that is adjusted to ensure convergence.

$$B_k = J(x)^T J(x) + I\lambda \approx \nabla^2 f(x_k)$$

A positive λ ensures that B_k is never non-singular and that the eigenvalues of this approximate Hessian are positive. When λ is small, this method becomes the Gauss-Newton method. If $\lambda \approx 0$ and $r_i(x) \approx 0$, this method resembles Newton's method. And large values of λ make this method mimic steepest-descent. The Levenberg-Marquardt method simply ties these three search strategies together with the addition of the parameter λ.

2.6 Summary

We have discussed several search algorithms, from a variety of disciplines, that are appropriate for solving nonlinear optimization problems. Each of these methods have a distinct set of heuristics that are used to explore the solution space and find more effective parameter values. These heuristics shape algorithm behavior, causing each method to respond differently to the features of a given problem. This creates a niche—a set of strengths and weaknesses—for each algorithm.

Gradient-based algorithms are extremely efficient local optimization strategies when gradient information is available. It is unlikely that a direct-search strategy will be competitive in this domain. In practice, however, the physical world that the objective represents may be quite complicated, and computing derivatives can be difficult, or even impossible, in many instances. And these methods will only find the local optimum that exists in the basin of attraction for which they are initially started.

Direct search methods were created, in part, because researchers were tired of the tedious nature of optimization using classic gradient methods. Direct-search methods allow practitioners to find quick and dirty solutions to a particular problem without the complexity of using a gradient method. Sometimes this is good enough.

Evolutionary algorithms—and in particular genetic algorithms—were developed with modality in mind. The deterministic search methods we have discussed, both gradient-based and pattern search, are best thought of as local search strategies. When employed on a multimodal surface, multiple restarts are the only heuristic that gives them a global perspective. Evolutionary algorithms are often thought of as being better global optimization techniques because they tend to sample more of the search space first, before isolating a particular region as the most effective.

Chapter 3

Comparing CMA-ES, CHC, and Pattern Search Using Diverse Benchmarks

Empirical studies are commonly used to understand algorithm behavior. As mentioned in the introduction, most empirical studies use a standard set of benchmark functions because they are readily available and the evaluation time is fast compared to many real-world applications. Often the goal when comparing different algorithms across these benchmark problems is to show that one particular algorithm, usually a novel one, is better than a set of existing and well know heuristic methods. But this approach frequently generates two complaints about evaluating evolutionary algorithms:

1. The test problems are too simple and easy.

2. There are not enough comparisons that include optimization algorithms used in the larger scientific community, such as the pattern search methods described in the previous chapter.

This chapter addresses these concerns in the following way. First we describe several of the synthetic test functions used to evaluate evolutionary algorithms and include a brief description of why these functions are believed to be difficult. We make this standard test suite more challenging by: 1) including a rotated version of each problem, 2) describing two new versions of the Griewangk function, and 3) introducing synthetic version of the "static correction problem" from geophysics. This problem is nonlinear, multimodal, scalable and it has the nice property that one can visualize the solution.

Then we compare CMA-ES, CHC, GPS, MADS, and local search on this set of problems. The comparisons are not meant as a competition. In fact, our results suggest that no one method is the best across all problems. This conclusion motivates the question: can these observed differences

in algorithm performance be related to problem features? We dissect our results based on a subset of the problem features described in the introduction. Our discussion suggests that the consistent differences in algorithm performance can be related back to problem features.

We focus our discussion on three specific features: separability, ridges, and global structure. These problem characteristics account for the majority of the discrepancies in performance we observe between the different algorithms we tested. One of the surprising results of this chapter is how much a simple rotation of the search space can change the performance of some algorithms. We review the literature on algorithms and their relationship to problem separability in this chapter. In the coming chapters, we explore more deeply how ridges and global structure affect some of the algorithms presented here.

We include a rotated and shifted version of most benchmark problems because many of the test functions described here are *separable*. Since separability creates an unrealistic surface for algorithms that exploit this feature, we review a method for altering such functions so that they are non-separable. This also ensures that the problems are no longer symmetric and that the global optimum is not located at some fortuitous location in the search space. All of our problems also have bound constraints on the input domain; thus, an algorithm that is invariant under rotation (e.g. CMA-ES) will *not* necessarily produce the same results in expectation on the rotated version.

3.1 Benchmark Test Problems

Below is a description of the most frequently used test functions in the evolutionary algorithm community. Table 3.1 lists the corresponding equations.

Sphere : This function is a simple convex unimodal surface. It is separable, symmetric,equally scaled in all dimensions, and invariant under rotation, arguably making it the easiest test function. Much of the theory developed in the evolution strategy community is connect with the sphere function. Bäck points out the importance of functions like the sphere: "..., before we can expect an algorithm to be successful in the case of hard problems, it has to demonstrate that it does not fail to work on simple problems" [8].

Ellipse : Similar to the sphere in all its properties except that it is scaled differently in each dimen-

sion. This creates a ridge axis along the dimension that has the smallest weight.

Rosenbrock : Rosenbrock's "banana function" is a well-known and difficult *ridge* function [81]. Rosenbrock notes, "Computer runs were started from $x_1 = -1.2$, $x_2 = 1$, so that the current point had to descend into the valley, and then follow it around its curve to the point $(1, 1)$"[81]. Voigt has recently argued that the curvature of the ridge, not the scale, is the primary feature that make the high dimensional Rosenbrock problem difficult [98].

Rastrigin : The Rastrigin function [80] is created by perturbing a sphere with a cosine term to create a highly multimodal surface that has a unimodal underlying shape. The local optima of this function are also located on the integer coordinate axes, making this function easier for algorithms that exploit symmetric topology (discussed below). Mühlenbein, Shomisch, and Born [67] are responsible for the modified version listed in Table 3.1.

Ackley : The Ackley function [1] generalized by Bäck [10] has a large number of local optima covering its simple underlying surface structure. Ackley suggests that "an iterated hillclimber will get stuck on one of the hilltops most of the time" [1]. At the same time, however, Ackley acknowledges that a search strategy with a larger neighborhood will likely be able to "see the higher lands hidden by the intervening valley", and therefore, these smaller traps are less of a damaging feature.

Griewangk : The Griewangk function also oscillates on a sphere surface that becomes more smooth, and therefore easier, in high dimensions [109].

Bohachevsky : Bohachevsky [17] proposed a test function with "many local optima" as a way to test his generalized simulated annealing algorithm. The function listed in Table 3.1 is one instance of the general class of functions proposed by Bohachevsky.

Schaffer : Schaffer *et al.* [86] originally proposed two test functions. Both functions are multimodal with different barrier heights between the local optima. Schaffer *et al.* state that their functions were designed to be difficult for search by simulated annealing [86]. The one listed in Table 3.1 is the most common.

Schwefel : Schwefel's function is actually one of several described by Hans-Paul Schwefel and with which his name is unofficially connected. This particular function is based on problem 2.3, page 328 in the 1995 edition of his book "Evolution and Optimum Seeking" [88]. This problem is a separable multimodal function where the second best local optima in the search space is distant from the global solution, which occurs near the bounds of the search space. Schwefel found that, "only the multi-member (e.g. $\mu > 1$) evolution strategy converged to the apparent global minimum; all other methods only converged to the first (nearest) local minimum" [88].

F101 and Rana : Whitley *et al.* noticed that most of the benchmark test functions were separable and symmetric. In response to this, they constructed test functions that were nonseparable, nonsymmetric, highly multimodal, and had a unique global solution along the diagonal of the two dimensional surface [109]. The first function, F101, was designed to be similar to Schwefel's problem, but nonseparable. The Rana function has ridge features and is highly multimodal. This is the rightmost graphic in Figure 3.1.

F8F2 : This function was also introduced by Whitley *et al.*. It is a nonseparable composite function that passes the result from Rosenbrock's function (traditionally called F2) to Griewangk function (traditionally called F8); this adds numerous local optima to Rosenbrock's original surface. A slice of the 2D surface (leftmost graphic) in Figure 3.1 highlights the additional local optima that form on Rosenbrock's problem.

The primary complaints with this set of benchmark test functions is that most of the functions are too simple and do not represent realistic problems. We address these concerns in three ways. First, all of the above multimodal functions, except *F101, Rana, and F8F2*, are separable. Separable problems contain no non-linear interactions between the parameters of the objective function. Although some problems can be separable and still have some degree of non-linearity (e.g. $f(\vec{x}) = x_1 \cdot x_2 + x_3$), these problems, as noticed by Whitley *et al.* [109], contained no non-linear interactions between parameters. The implication here is that these functions can be easily solved by

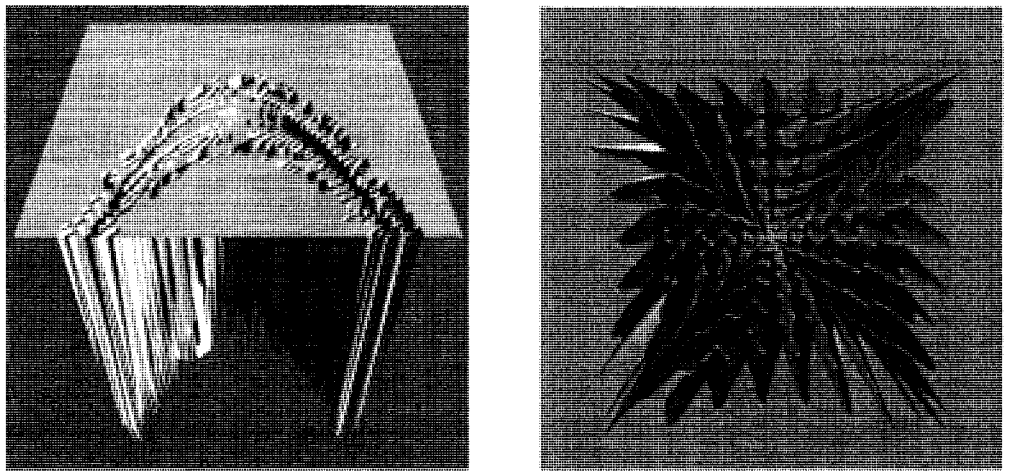


Figure 3.1: Two-dimensional surface slices of the F2F8 (left) and Rana function (right).

searching for the optimal solution separately in each dimension. As we will see in the next section, several algorithms exploit this characteristic.

Second, the Griewangk function exemplifies another concern within this test suite; it actually gets smoother, and therefore, easier as the dimensionality of the problem increases. We propose two modifications to this function that allow problem difficulty to scale well with dimensionality.

Finally, there is a general concern that test problems formed by mathematical equations may lack the problem structure found in more complex and realistic applications. In this section, we model a synthetic version of a problem from geophysics.

3.1.1 Separability

Both Salomon [83] and Whitley et al. [109] noted about 10 years ago that most functions used to evaluate evolutionary algorithms are *separable* and can easily be solved to optimality by independently searching each dimension of the search space. Whitley *et al.* proposed some new nonseparable test functions (F101, Rana, and F8F2). Salomon created new nonseparable problems by rotating some of the existing test functions. This rotation make the problems nonlinear and also breaks symmetry in the search space that can make problems much easier to optimize.

Salomon [83] found the Breeder Genetic Algorithm [66] performed poorly after rotation and he concluded from this that recombination was of limited value as a search operator on surfaces where there was a high interaction between the parameters of the objective function. Part of the problem is that the Breeder Genetic Algorithm uses the coordinate axis as basis for recombination. The

Name	Function	Domain
Sphere	$\sum_{i=1}^{N} x_i^2$	$[-5.120, 5.110]$
Ellipsoid	$\sum_{i=1}^{N} (100^{\frac{i-1}{n-1}} x_i)^2$	$[-5.120, 5.110]$
Rosenbrock	$\sum_{i=1}^{N-1} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$	$[-2.408, 2.408]$
Schwefel	$\sum_{i=1}^{N} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$[-512.0, 511.0]$
Rastrigin	$10 \cdot N + \sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i))$	$[-5.120, 5.110]$
Schaffer	$\sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{\frac{1}{4}}) \cdot \left[\sin^2\left(50 \cdot (x_i^2 + x_{i+1}^2)^{0.1}\right) + 1.0\right]$	$[-10.24, 10.23]$
Bohachevsky	$\sum_{i=1}^{N-1} (x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_{i+1}) + 0.7)$	$[-1.024, 1.023]$
Griewank	$\frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-512.0, 511.0]$
Ackley	$-20 \cdot \exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N} \cos(2\pi x_i)\right) + 20 + e$	$[-10.24, 10.23]$
F101	$\sum_{i=1}^{N-1} -x_i \sin(\sqrt{\lvert x_i - y \rvert}) - y \sin(\sqrt{\lvert y + x_i/2 \rvert})$, with $y = x_{i+1} + 47$	$[-512.0, 511.0]$
Rana	$\sum_{i=1}^{N-1} x_i \sin(\sqrt{\lvert -x_i + y \rvert}) \cos(\sqrt{\lvert x_i + y \rvert}) + y\cos(\sqrt{\lvert -x_i + y \rvert}) \sin(\sqrt{\lvert x_i + y \rvert})$, with $y = x_{i+1} + 1$	$[-512.0, 511.0]$
F8F2	$\sum_{i=1}^{N-1} \frac{\text{Rosenbrock}(x_i, x_{i+1})^2}{4000} - \prod_{i=1}^{N} cos\left(\frac{\text{Rosenbrock}(x_i, x_{i+1})}{\sqrt{i}}\right) + 1$	$[-2.048, 2.047]$

Table 3.1: Common synthetic test functions.

question is: can we change the basis we are using and have a more effective genetic recombination operator?

In the mid 1990's, Kazadi [49] motivated a new representation for genetic algorithms called a conjugate schema. Like Salomon, Kazadi asserts that sometimes crossover disrupts the efficiency of a real-valued genetic algorithm by producing low fitness offspring. As a solution, Kazadi proposed

finding a new basis, called a conjugate schema, that minimizes the functional dependencies between the parameters of the objective. This new basis creates local separability in the objective function near the current point and, hopefully, allows the vectors of the genetic algorithm to cross with a higher likelihood of strong children. In other words, instead of forcing the offspring to align with the coordinate axis, as is the case of most genetic algorithms, the rotated conjugate basis aligns with local features and creates offspring that are more likely to fall on these higher fitness areas. Kazadi uses the eigenvectors of the absolute Hessian matrix to find an optimal basis.

One practical problem here is that Hessians require twice differential functions, and can be difficult to compute. This certainly takes away some of the appeal of using a "black box" evolutionary algorithm. Kazadi explains a computationally expensive method for rotating a coordinate basis. Unfortunately, the rotated basis method only generates orthogonal basis, and therefore, cannot necessarily guarantee *conjugate* schema. Kazadi also points out that conjugate schema continually change depending on the location in the search space. This implies that a good basis for one point may not be appropriate for another point. Recomputing a new basis for different intervals of the search, as proposed by the author, doesn't seems to fix the problem until the population starts to converges to the same neighborhood.

Wyatt and Lipson [111] arrive at similar conclusions from a different perspective. Their work focuses on *genetic linkage*, which is a measure of the correlation between functional dependency and gene location. Ideally, representations with high genetic linkage will push separable parts of the problem together, and these parts will become useful building blocks for genetic search. In order to find an optimal gene ordering, Wyatt and Lipson also use the eigenvectors of the Hessian around the current best point, and uses this ordering for the entire population. This is is not the same as finding a more efficient rotational basis. Instead, Wyatt and Lipson are using the Hessian to determine the degree to which the parameters of a problem interact, and then use this knowledge to find an ordering of the parameters that is less disruptive under a given crossover operator.

The computational cost of the Hessian is still a practical problem. Wyatt and Lipson also briefly discuss higher-order transformations that could potentially transform landscapes that are non-linear into linearly separable problems. Like the method of *conjugate schema*, different representations are better in different parts of the search space, and again, using the representation around the current

best only seems effective as the population approaches a similar optima.

Although these ideas may be useful in addressing the shortcomings of algorithms that use the coordinate axis as its search basis, but they are far from being a practical solution. This adds to the growing concern that discrete crossover is bias toward the coordinate axis. However, using a real-valued representation does not necessarily solve this problem; as previously mentioned, Salomon [83] found that the Breeder Genetic Algorithm [12], which uses the real-valued representation, was highly dependent on the degree to which a function is separable.

Real-valued genetic algorithms require reproductive operators that act on real-valued vectors. One way of recombining two parents is simply to average their positions in a weighted or unweighted form [110]. This is identical to the recombination operator used in an evolution strategy. Figure 3.2 shows an example of this type of *Linear* recombination. Eschelman and Schaffer propose the "blend crossover" operator, denoted *BLX-*α [31], which is a generalization of Radcliffe's "flat" recombination operator. BLX-α produces an offspring based on an interval that extends α times past either parent. In two dimensions, the parents define a box that has a border of width $\alpha \cdot I_i$, where I_i is the length of the interval in dimension i. Figure 3.2 shows this two dimensional example. Offspring are chosen uniformly such that they fall within the defined boundary.

Eschelman and Schaffer compared their BLX-α operator against the discrete version of CHC that uses bit-strings and were left with mixed results; sometimes BLX-α was more effective, sometimes there was no difference between the two algorithms, and at times, CHC used fewer evaluations to find more effective solutions. Most of the problems they tested each algorithm on are separable.

Ono and Kobayashi argue that the BLX-α operator creates children that do not always inherit the parents strong characteristics [71]. They continue, " This often occurs and makes the search inefficient when the parents are on a valley or ridge that is not parallel to the axes of the coordinate system". As a solution, Ono and Kobayashi offer a *Unimodal Normal Distribution Crossover* operator, denoted UNDX, that creates offspring based on a distribution defined by three parents. The normal distribution between the first two parents and is proportional to the Euclidean distance between them. All other $n - 1$ standard deviations are based on the orthogonal distance from the third parent to the axis that connects the first two. Figure 3.2 also shows an example of this type of crossover.

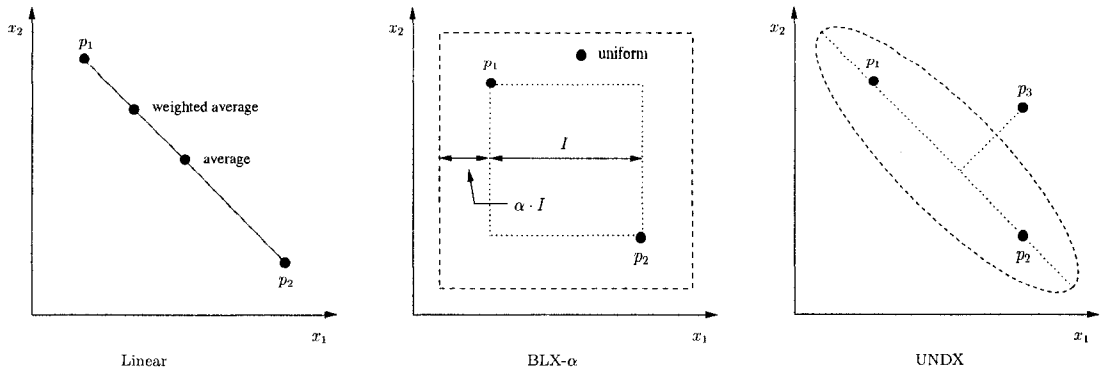


Figure 3.2: Real-valued crossover. The left-most figure shows a simple weighted average of the two parents. The middle figure is the *blend crossover* operator. Individuals are created anywhere in the box, including the area that extends past each parent. Finally, the right-most figure is UNDX mutation distribution defined by the three individuals $p1$, $p2$, and $p3$.

One concern here is that this crossover operator is creating offspring in high dimensional space based on a distribution that is being modeled by very little information (using only three individuals, regardless of dimensionality). Although the authors claim that the offspring inherit more of the parents characteristics, this is really based on the assumption that strong individual properties occur in the neighborhood of a line segment between two parents. This is clearly different from learning a better representation. It is rather a heuristic for creating offspring in a way that is not bias toward the coordinate axis—two very different concepts.

The evolution strategy community has often focused on mutation as the main operator for creating variation in the population. Some mutation operators are invariant to rotations of the search space. Salomon concluded that, unlike the breeder genetic algorithm, the simple self-adaptive evolution strategy is invariant under rotation, and therefore, not affected by parameter interaction [83]. Similarly, CMA-ES is also invariant to rotations of the search space. We expect these two evolution strategies to perform similarly regardless of how strongly the parameters of a given problem interact.

Rotating Test Functions

Many of the previously mentioned empirical studies determine an algorithm's rotational bias by comparing its performance on a separable function with that of the same function rotated by an orthogonal transformation matrix [71; 83; 43]. Rudolph [82] showed how to create a random or-

thogonal rotation matrix. To the best of our knowledge, Salomon [83] was the first to use this method to rotate separable functions.

Rotating the search space can change the degree to which the parameters of a given test function interact. Rotation is commonly done by first creating an orthogonal rotation matrix $\mathbf{M}$, and then transforming the input parameters $\vec{x}$ before evaluating the individual. Specifically, if $f(\vec{x})$ is the objective function then the individual that corresponds to the point $\vec{x}$ is assigned the fitness of $f(\mathbf{M}\vec{x})$. We create an orthogonal matrix in a method similar to that of Salomon [83]. In order to increase reproducibility and control the degree of parameter interaction, we use a constant rotation angle of $\alpha = 22.5$ for every dimension.

Rotation also breaks the symmetry that exist around the coordinate axis. One drawback of using symmetric functions for evaluation is their potential bias toward search neighborhoods that use a bit-encoding. For example, standard reflective Gray code always creates a symmetric neighbor reflected about the origin. On functions like the two dimensional Rosenbrock, finding the local optima at (1,-1) implies one of the neighbors for the next search will be the global optima located at (1,1). Eiben and Bäck catalog multimodal functions differently based on the symmetry of the local optima [29]. In this chapter, we translated each function by 5% of the domain to reduce symmetry that exists around the origin of the search space.

Comparing an algorithm's performance on a rotated and unrotated problem is straight forward when the optimal solution is near the center of the bound search space. In this case, detecting a performance difference is easy because the optimal solution should be the same. Any discrepancy is due to rotational effects. However, when the optimal solutions lie near the boundaries of the search space, the optimal solutions will not necessarily be the same for the rotated and unrotated surfaces if the domain is constrained. This means search algorithms will not necessarily produce the same results for the rotated and unrotated problem.

Rotation is one way of making a separable test function more realistic. The remainder of this section describe two additional contributions to our test suite: the revised Griewangk functions and a synthetic static correction problem.

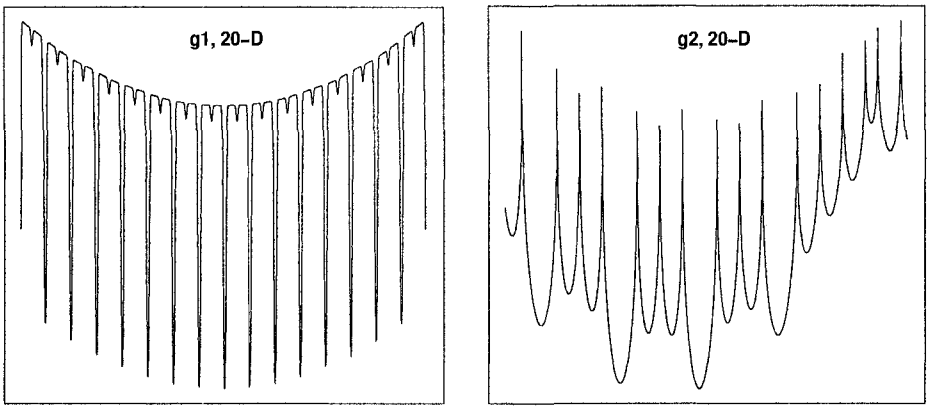


Figure 3.3: Diagonal slices of G1 and G2 in 20 dimensions. While the original Griewangk function becomes smooth and easier, these modified problems retain the modality that Griewangk intended in higher dimensions.

3.1.2 Revisiting Griewangk's function

The original Griewangk function is as follows [109]:

$$f(x_i|_{\,i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (cos(x_i/\sqrt{i}))$$

The global optimum is at the origin. The search space is bowl shaped and local optima are created over the bowl through the oscillation of a cosine function. The problem is, the range of a cosine function is $[-1, 1]$ and as the number of dimensions increase, multiplication of the cosines destroys the local optima. Thus, higher dimensional versions of this problem become smooth with a strong global optimum [109].

We used two modifications of the original Griewangk function denoted `G1` and `G2` that include a scaling factor that is added to the summation term to stabilize the function range across dimensions [91]. In addition, the output from a cosine function is translated and/or scaled to offset the effect of multiplication. For `G1` and `G2` the global optimum does not have a consistent value as the number of dimensions is varied. The new forms of Griewangk are given in Table 3.2. `G1` is characterized by taking a logarithm of the product term. Figure 3.3 shows slices of `G1` and `G2` at 20D. The optima are now rather wide with thin barriers separating one from another. On `G2` the search space also loses its symmetry due to a phase shift.

Name	Function	domain
G1	$\sum_{i=1}^{N} \frac{x_i 2}{4000N} - 1.5^{N/4} \left[\prod_{i=1}^{N} \sqrt{cos(x_i/N + i) + 1.0} \right]^{1/4}$	$[-512.0, 511.0]$
G2	$\sum_{i=1}^{N} \frac{x_i^2}{4000N} - -1.5^{N/4} \left[\prod_{i=1}^{N} \sqrt{cos(x_i/N + i) + 1.0} \right]^{1/4}$	$[-512.0, 511.0]$

Table 3.2: New Griewangk functions.

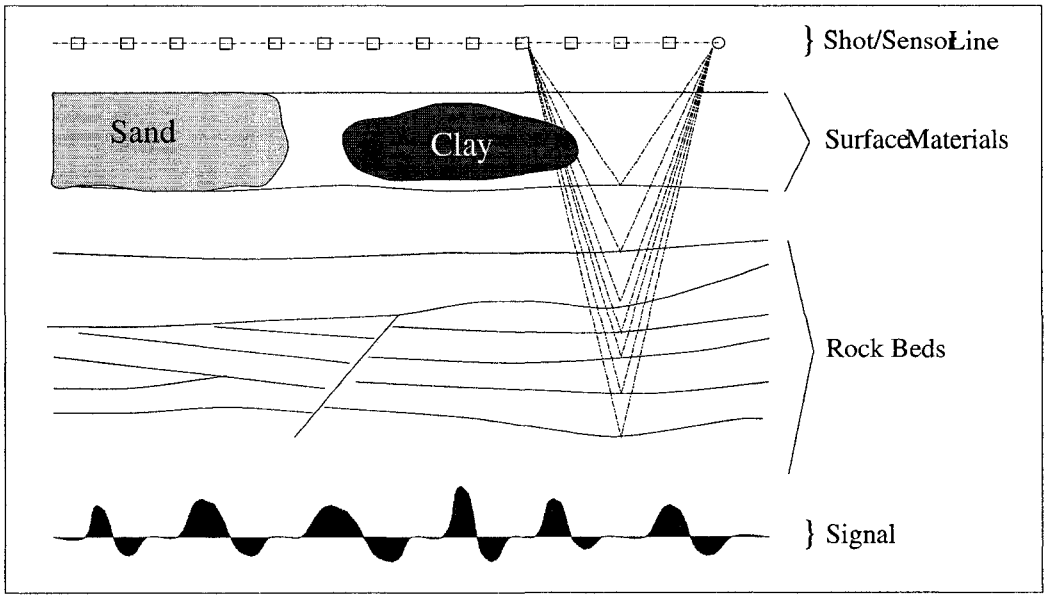


Figure 3.4: Sensor Shot Signals.

3.1.3 Synthetic Static Corrections for Seismic Surveys

All the test functions previously mentioned are easy to disseminate and have fast evaluation times because they are simple mathematical equations. We propose a synthetic test function here that sacrifices some of this convenience, but in turn, adds diversity the problem structures that exists within our test suite.

There are many difficult optimization problems in geophysics. One such problem is the "static corrections" problem. Seismic reflection surveys are used to construct subsurface images of geologic strata. The images are distorted by variation in surface materials and ambient noise [63].

Seismic reflection surveys are performed by setting off a detonation (shot) and then recording the resulting signals at a series of sensor sites. Figure 3.4 illustrates the reflection signals that a single sensor (□) would collect for a single shot (○) as well as the waveform corresponding to the

signal. Each of the sensors along the sensor-line would collect a similar set of reflection signals for this shot. (The data can be filtered and refined using common midpoint gathers of the signals. We ignore this detail in our discussions.)

Differences in the materials at the earth's surface affect the arrival time of seismic signals. Loose materials, such as sand, slow down signal arrival, whereas packed materials, such as clay, speed up signal arrivals. Given a collection of signals, a "static correction" can be made that corrects for differences in arrival time; then the collection of signals can be aligned to produce an accurate image of the subsurface strata. The optimization problem is to find a *static correction* for each signal such that it correctly aligns strata to reveal subsurface geology. Misalignment of signals can result in numerous local optima, and conventional optimization methods can easily be trapped in these sub-optimal regions. These methods are unacceptable when there exists severe surface heterogeneities or a high level of ambient noise.

We introduce a synthetic static correction problem as a test function. The synthetic problem consists of a base matrix, which defines the structure of the simulated geologic strata. The base matrix M has i rows and j columns, where i indexes the i^{th} strata and j indexes the j^{th} signal. Entry $M(i,j)$ stores the depth of strata i as observed in signal j (we can assume depth=time). For example, consider the following $3x5$ base matrix:

$$\begin{pmatrix} -100 & -100 & -100 & -100 & -100 \\ -375 & -350 & -300 & -275 & -250 \\ -450 & -525 & -550 & -525 & -450 \end{pmatrix}$$

To simulate near surface variation of the earth we add a nonuniform **profile**, denoted p, to the problem. The profile has length equal to the problem dimension. Element p_j in the profile is added to each element of column j of matrix M: this effectively shifts signal j by p_j units and creates a new matrix $\mathcal{M}$. The optimization problem is to search for and retrieve vector p given the shifted matrix $\mathcal{M}$. Finding p will undo the shifts and restore the original matrix M, and thereby align the geological strata.

Our simplified version of static correction assumes that the strata in M are structured and well behaved, while the shifts in matrix $\mathcal{M}$ are not regular. Fitness is usually calculated by computing the cross-correlation (e.g. the dot product) between all pairs of shifted signals. In the current experiments, we use simplified signals, such that the vector is 0 except when near the location of

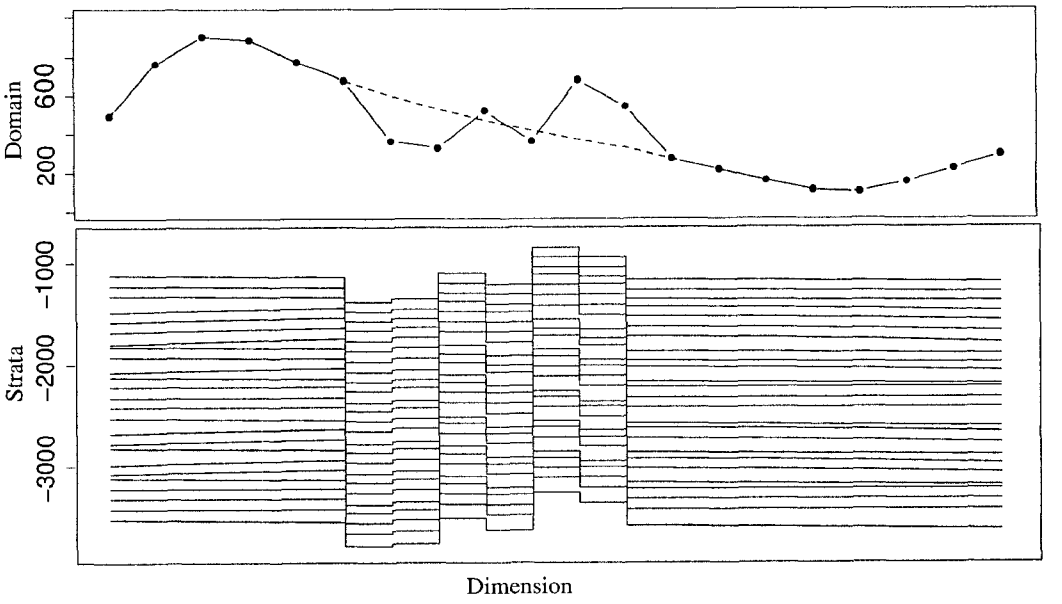


Figure 3.5: The synthetic static corrections problem. In the top figure, signals are indicated by column slices of the figure, and the known and proposed arrival times are shown across the top of the figure. If the solution is locally optima, moving any signal up or down (i.e., adjusting its arrival time in the Domain) will result in a poorer alignment of Strata.

a strata. Within a distance of delta of the strata, the vector is 1. When the total depth (length of the vectors) is greater than the number of signals (denoted by N), evaluation time is greater than $O(N^3)$. In our simplified model the dot-product is the same as overlap and we do not actually have to construct signals, which allows for evaluation in $O(N^2)$ time. If one wishes to model ambient noise the full evaluation must be used.

Figure 3.5 shows a 20 dimensional problem, including the simple profile vector, which does not have any discontinuities (upper graphic), and the deviation of the current solution from the profile (lower graphic). The signals are shown as offset column slices. Real static correction problems generally have 100's of variables.

This problem has several features: it is scalable, it can be made noisy, and more complex geology and signals can be used to make the problem more difficult; yet, it is easy to visualize misalignments in the solution. There are also several local optima that form from non-optimal alignments of the strata. Figure 3.6 shows a surface of the first two dimensions of a 20 dimensional problem with a delta of $+/-50$ units. The local optima in the surfaces are created from non-optimal strata alignments.

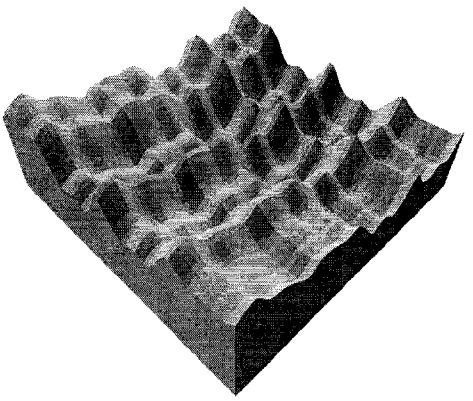


Figure 3.6: This figure shows there are many locally optimal alignments in a 2D slice of the search space.

3.2 Empirical Results

This section compares CHC, CMA-ES, GPS, MADS, and local search on a selection of 20 dimensional benchmark problems from the previous section. In order to keep the analysis here contained and manageable, not every benchmark test function is presented. Instead, we chose functions based on feature diversity and perceived difficulty. For example, we did not include the Rastrigin or Ackley functions in this section because we already have several test problems that have a unimodal bowl-like global structure that is perturbed by local optima. Including these additional test functions would not add feature diversity to the test suite.

Local search and CHC were run using 10-bits and 20-bits of precision. CHC used a population size of 50. We ran CMA-ES with rank-μ-updates and a population size of 200 and 500 [41]. These parameter settings are We distinguish each algorithm based on its parameters; CHC-10, CHC-20, LS-10, LS-20, CMA-200, CMA-500. The Pattern Search algorithms are MADS-2n, MADS(n+1), GPS-2n, GPS(n+1) and Unif(n+1), as described in chapter 2. Each algorithm was run for 50 trials; restarts ensured that each trial ran for exactly 100,000 evaluations. The results are presented in Figures 3.7, 3.8, and 3.9. Since we are minimizing each function, a lower value is a more effective solution.

The "big picture" observation we make is that no single algorithm is best across all problems. Sometimes there is a clear winner and other times there are no significant differences between the algorithms. But overall, there is no one algorithm that dominates all the others. The question we

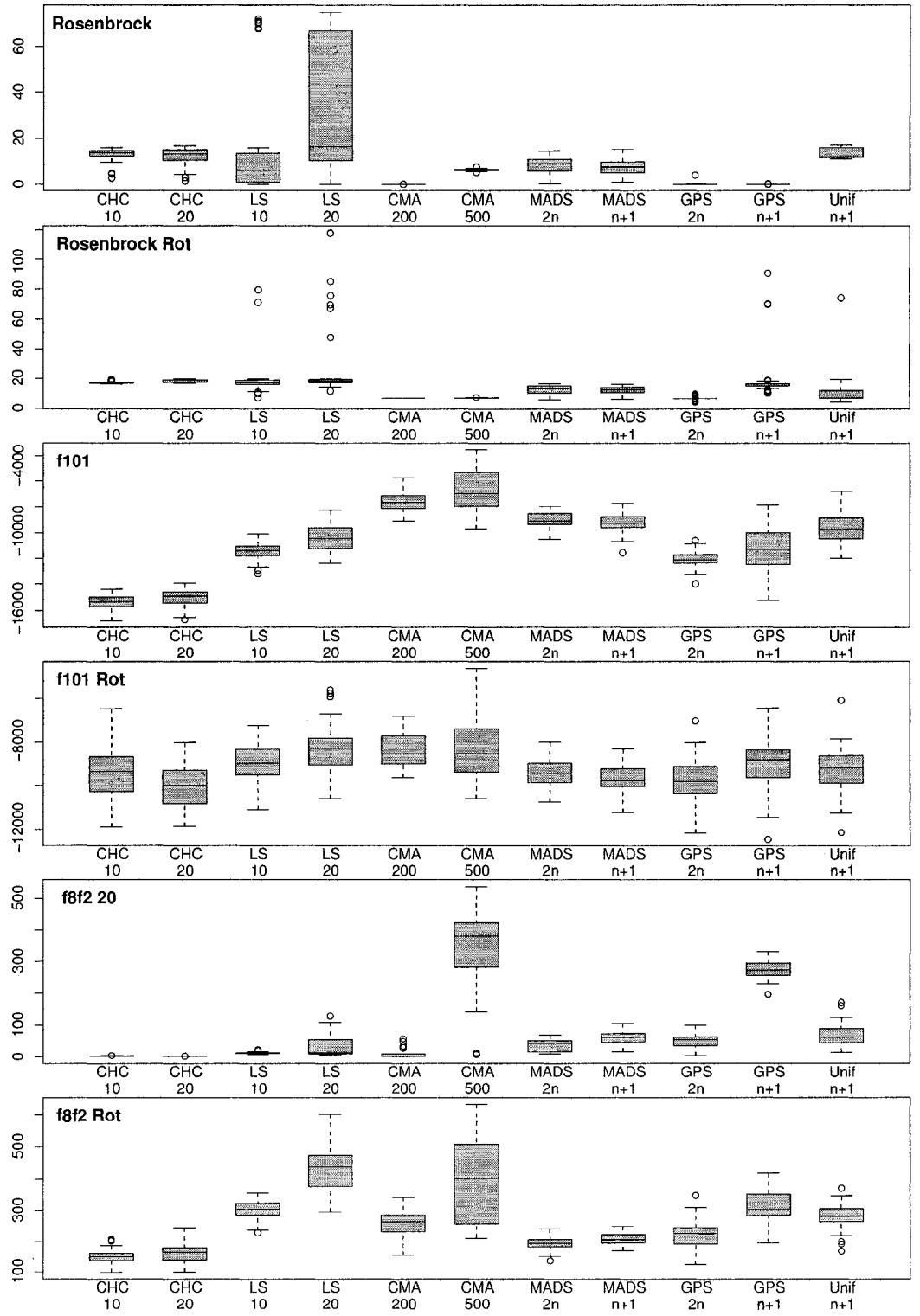


Figure 3.7: The midline in the gray box is the median, while the gray box represents 25 percent above and below the median. The bars outside the gray box generally represent the max and min values, except when there are outliers, which are shown as small circles. The y-axis represents the value of the evaluation function.

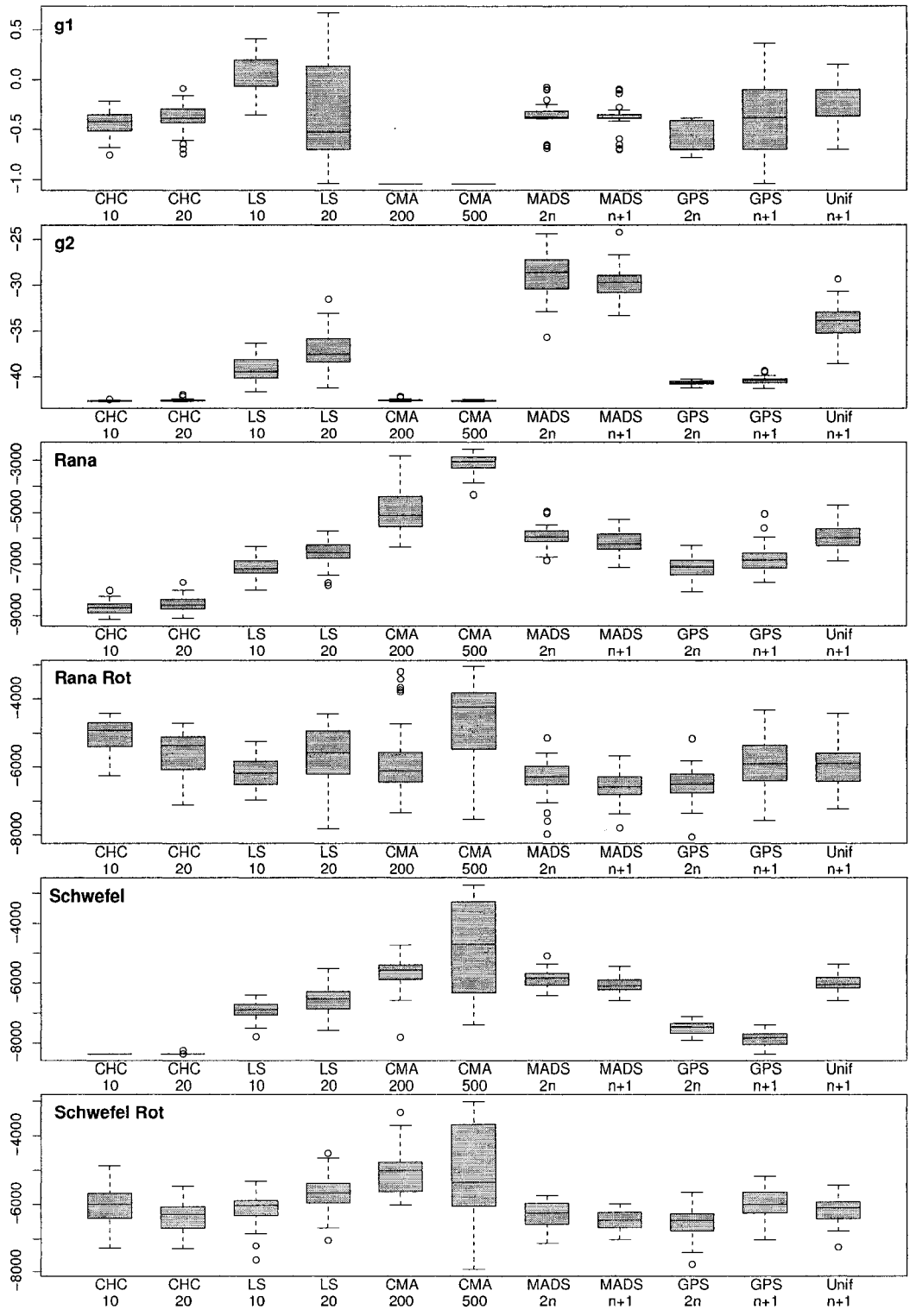


Figure 3.8: The midline in the gray box is the median, while the gray box represents 25 percent above and below the median. The bars outside the gray box generally represent the max and min values, except when there are outliers, which are shown as small circles. The y-axis represents the value of the evaluation function.

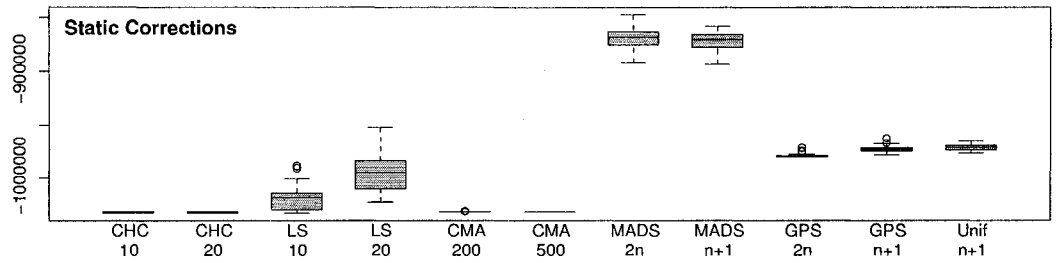


Figure 3.9: The static correction problem. The midline in the gray box is the median, while the gray box represents 25 percent of the sample above and below the median. The bars outside the gray box generally represent the max and min values, except when there are outliers, which are shown as small circles. The y-axis represents the value of the evaluation function.

explore here is: can we explain algorithm differences in terms of how well each algorithm addresses the features of a particular problem? We highlight three specific points with respect to the empirical differences we observe and then discuss how these differences relate to algorithm heuristics and problem feature interaction. The key points are:

1. **Separability**: Rotating the functions generally reduces the observed differences in algorithm performance. In particular, this occurs on the following functions: F101, F8F2, Rana, and Schwefel, all of which are located in Figures 3.7 and 3.8.

2. **Ridges**: Rosenbrock's ridge function (Figure 3.7), both rotated and unrotated, is difficult for most algorithms.

3. **Global Structure**: CMA-ES does well on those problems where the best local optima are clustered together. This includes the Static Corrections problem (Figure 3.9) and variants of the Griewangk function (Figure 3.8). On functions that lacked this "global structure", the performance advantage of CMA-ES was much less pronounced.

3.2.1 Revisiting Separability

The first observation we address is how a simple rotation of the search space impacts algorithm performance. Without rotation, CHC is clearly the most effective algorithm on the multimodal functions we tested. However, on the rotated versions of F101, Rana, the F8F2 composite function, and the Schwefel function, there is much less difference in performance between the algorithms being compared. In particular, the performance advantage of CHC often disappears when these

functions are rotated.

CHC and Local Search use a bit representation and appear very sensitive to both search space symmetry and its alignment with the coordinate axes. We already mentioned Salomon [83] conclusions with respect to recombination; he suggests that this search operator is of limited value on surfaces where there exists a high interaction between the parameters of the objective function. But this limitation does not extent to all genetic algorithms that use crossover. For example, CHC creates variation in the population from one generation to the next by using a variant of uniform crossover. This effectively changes many parameters at once, and the changes may be local or global. Although CHC does use a fixed coordinate system, our results suggest that Salomon's conclusion is too sweeping. We found the CHC to be highly competitive on F8F2 and F101 and Schwefel even after the functions are rotated. The results of this section simply support the well documented observation that some algorithms performance may be overly inflated because they exploit problem separability and symmetry.

3.2.2 Ridges

We also observe that Rosenbrock's function is difficult for most algorithms. As previously mentioned, Rosenbrock's function has a long narrow ridge that leads to its optimal solution. On the original version, CMA-ES with a population size of $\lambda = 200$ is the only evolutionary algorithm that solves this problem. CMA-ES with the higher population size of $\lambda = 500$ did not converge to the optimal solution because it did not have enough generations per trial (e.g. the population size is too large for this problem).

Most algorithms did better on the rotated version. This is because the rotation aligns the "banana" (ridge) more closely with the search space coordinates. Still, CHC and local search perform poorly on both the rotated and unrotated versions. GPS(n+1) performed better on the unrotated version of this problem. This is because the base pattern that the GPS "simplex" uses aligns more closely with the ridge on the original version. This coincidence does not imply that the pattern search methods are capable of exploiting ridge structure.

This example problem highlights the fact that the ridge feature is difficult for heuristic search. At the same time, there is surprisingly very little references made to this feature within the evolutionary

algorithm community. For example, both Kazadi's and Wyatt and Lipson's ideas of learning a better rotational basis are really focused how parameter interaction impacts crossover. These proposed ideas may indeed be useful, especially with respect to separability, but these studies take a very narrow view of how the representation is being changed. Neither of these studies were concerned with mutation or the direction of the gradient. On the other hand, the evolution strategy community, in particular the CMA-ES algorithm, has focused more on rotations and concerns about the search direction and step-size.

There are still open questions within the evolution strategy community with respect to ridge functions. The theoretical and empirical behavior of an evolution strategies using *constant mutation strength* and a single global step-size is well documented on ridge functions [76; 74; 75; 14; 5]. Beyer points out that the results derived using a constant mutation strength can serve as a performance benchmark for *self-adaptation* [14]. That is, if self-adaptation is working as expected, its performance should be close to that of the evolution strategy that does not adapt. Unfortunately, self-adaption fails on ridge functions, but it is unclear why.

This chapter highlights the need to understand how ridges impact search and why some heuristics fail to overcome this barrier. In the next chapter, we discuss how *local search* and the *self-adaptation evolution strategy* are affected by ridges.

3.2.3 Global Structure and Big Valley

Lastly, we point out that CMA-ES does well on those problems where the best local optima are clustered together in a "big valley" or single-funnel. This includes the Static Corrections problem and variants of the Griewangk function. On functions that lacked this predictable global structure, the performance advantage of CMA-ES was much less pronounced. In fact, CHC performed best on the functions that do not have a big valley structure. Later in this thesis, we show that CMA-ES is also less efficient on problems that lack a simple global structure. We use a *dispersion metric* to predict how well CMA-ES will perform and describe why it is inefficient on some problem instances.

Like the ridge problem, there is almost no mention of how global structure impacts evolutionary algorithm performance, or what the most effective and efficient strategies are for searching through

landscapes that do not have a strong attraction towards the global optimum. In fact, there are only suggestions that global structure may play a larger role in the success of global optimization attempts. Kern *et al.* assert that "For the CMA-ES, functions are hard to solve where the attractor volume of the global optimum is small, and an overall topology pointing to the global optimum is missing" [50]. This is an important direction of research because real problems are emerging where the global optimum is not located in the largest volume of the search space. Chapter 7 is devoted to creating test functions that have more than a single-funnel (big valley) landscape, and discussing how this impacts evolutionary search.

3.3 Summary

We have introduced the standard set of test problems used to evaluate evolutionary algorithms. To make these problems more realistic, we discussed how rotation is used to transform separable and symmetric surfaces into landscapes that have less symmetry and a higher degree of parameter interaction. Two variations of Griewangk's function are introduced that prevent local optima from being destroyed as the dimensionality increases. We also introduce a synthetic version of the "static corrections" problem. Together, these modifications create a more challenging and realistic benchmark suite.

When we compare a diverse selection of algorithms on this relatively difficult set of test problems, we find that no single algorithm dominates the others, and different algorithms have better performance on different types of problems. Furthermore, these differences can be related to problem features.

Yet these kinds of differences are rarely reported in the literature; especially how these differences relate to problem features. Perhaps the main significance of this chapter is that the evolutionary optimization community still has not been able to develop a clear understanding of how different algorithms exploit different types of problem features. The remaining chapters of this thesis are dedicated towards exploring and understanding this gap as it relates to ridges and global structure.

Chapter 4

Understanding Local Search and Self-Adaptation on Ridges Functions

Ridges in a search space are a difficult barrier for most search algorithms. In the previous chapter, we observed that several of the algorithms discussed so far performed poorly on Rosenbrock's ridge function. A ridge is essentially comprised of two characteristics: *direction* and *scale*. The difference in *scale* between the objective parameter values creates the ridge feature and the interaction of the parameters determine how the ridge is oriented in the search space. If the objective function is twice differentiable, the *condition number* of the Hessian, which we discuss shortly, is one way of quantifying the degree to which the ridge is poorly scaled.

The ridge characteristics, *scale* and *direction*, affects different algorithms in different ways. If an algorithm looks for improving moves by changing only one dimension at a time (e.g. a coordinate pattern), it will not see better points that fall between the neighborhood axis. This is the *direction* problem. A similar problem occurs for algorithms, such as GPS, that use a fixed pattern; search is blind to any ridge that is not aligned with its pattern.

The ridge *scale* also affects search. Isotropic distributions are invariant to rotations of the search space, meaning that the offspring are created in an unbiased search direction. But being invariant with respect to direction and being able to exploit ridge structures is not exactly the same [105]. The implication here is that problem *scale*, not the ridge orientation, is the primary characteristic that will affect performance.

In this chapter we explore *why* ridges are detrimental for local search and the self-adaptive evolution strategy. First, we show that local search can exhibit extremely slow convergence, or even

fail, on functions that display ridge structures. We explain how representation and precision account for this observed behavior. We also propose and explore a class of rotated representations; these can be based on Principal Components Analysis (PCA), or use the Gram-Schmidt orthogonalization method. Some algorithms, such as CMA-ES, already make use of similar rotated representations.

Second, several researchers have also noticed that the self-adaptive evolution strategy can be extremely inefficient on ridge functions because the global step-size becomes too small. On the parabolic ridge we conjecture that this step-size will stabilize when *selection* is unbiased towards larger or smaller step-sizes. On the *sharp ridge*, where the bias in selection is constant, the step-size will continue to decrease. We provide empirical evidence that supports our conjecture and accounts for self-adaptation's poor performance on ridges.

4.1 Background

Some of the other common benchmark test functions described in chapter 3 contain ridge features. The leftmost 2D illustration in Figure 4.1 is Rosenbrock's classic unimodal ridge that we have already discussed. The Rana function (middle 2D illustration) also contains ridge features. The rightmost cartoon shows the direction of the ridges that lead to the best regions of the search space.

The direct search community has been aware of the ridge problem since the early 1960's. In fact, Rosenbrock created his test function specifically to illustrate the weakness of methods which change only one variable at a time during search [81]. Rosenbrock showed that even some gradient methods move very slowly on this function because the direction of the gradient significantly changes at each time step.

Rosenbrock proposed a search method that uses the Gram-Schmidt orthogonalization algorithm to adapt the search coordinate system. Later the Nelder-Mead Simplex method was introduced [69], in part, to deal with this problem. These methods often compute a direction of improvement based on a sample of points; then, line-search type methods are often used to look for improving moves. In theory, these methods should be able to follow ridge structure *if* they select the correct direction. The potential disadvantage of these methods is that they heuristically compute a direction based on very little information.

The mathematical optimization community also understands why it is necessary to pay atten-

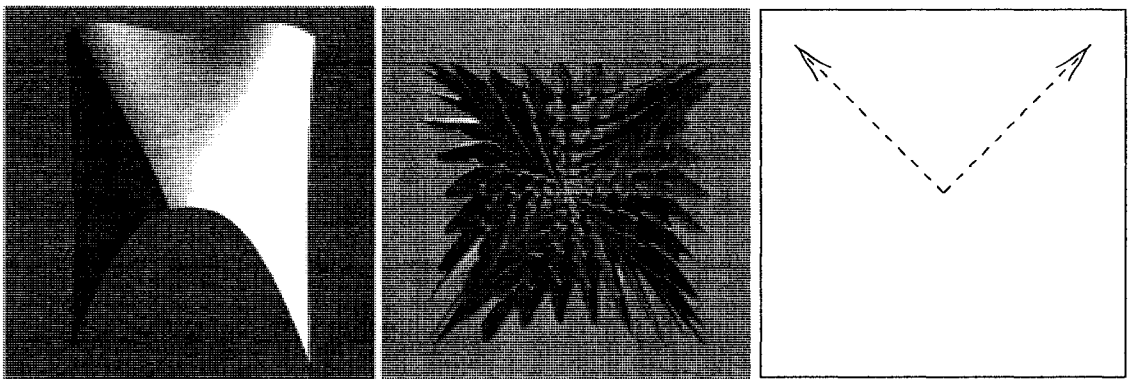


Figure 4.1: The leftmost figure is Rosenbrock's function. The middle figure is Rana's function. The rightmost figure is a cartoon showing the "ridges" that lead to the global optimum as well as other competitive local optima.

tion to algorithm performance on ridge features. About two hundred years after Newton's method was invented, Cauchy proposed a more simple optimization method: compute the gradient and then line-search in that direction iteratively until a local optimum is reached [68] . This is the method of *steepest-descent* as described in chapter 2. But following the gradient turns out to be an extremely inefficient way of optimizing a function that contains ridges. As previously mentioned, the efficiency of steepest-descent is directly proportional to how poorly-scaled the ridge is.

The *condition number* of an objective function's Hessian at a particular point is standard way of measuring the localized change rate in the objective function (e.g. its scale). The *condition number* is the ratio of the maximum and minimum singular values of the Hessian. Larger condition numbers correspond to problems that are *ill-conditioned*; the ridges that form in these landscapes have a larger difference in scale. Graphically, the maximum and minimum singular values correspond to a landscape projection that has the most dramatic ridge feature. The two quadratic surfaces in Figure 4.2 have different condition numbers. The rightmost graphic has a higher condition number, and therefore, causes steepest-descent to take more steps in order to reach the optimum.

The gradient methods that work well on ridge functions either 1) estimate a better search direction, as is the case for the *conjugate-gradient* method, or 2) they attempt to estimate the inverse Hessian and use this in Newton's method. The merits of finding the correct search direction is rather intuitive; instead of taking multiple steps in the same direction, why not get the direction correct the first time and just take one [90]? Estimating the inverse Hessian is effective because it transforms a quadratic ridge surface into a sphere. It then only takes one line search method in the direction of

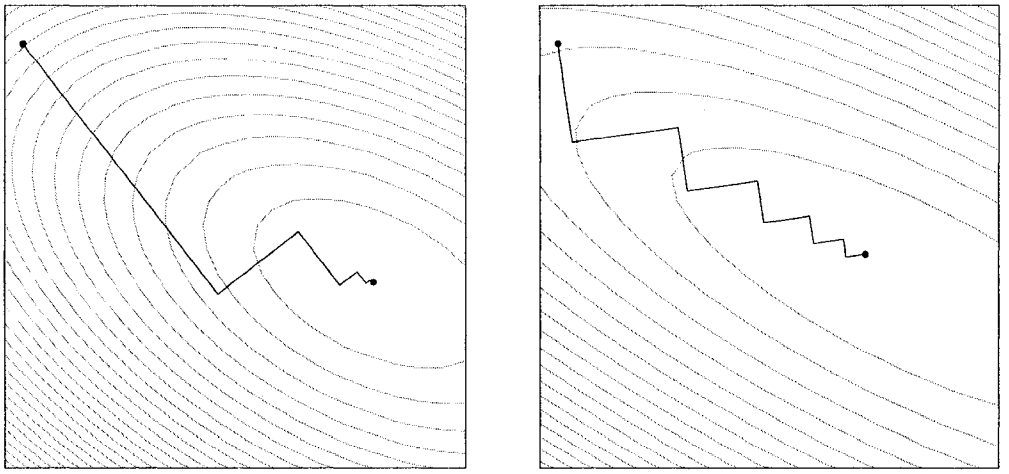


Figure 4.2: Steepest-descent on two different quadratic surfaces. More pronounced ridges in the search space (rightmost figure) cause steepest-descent to be less efficient because it must take more steps to reach the optimum.

the gradient to reach the optimum. In either case, the complexity cost per iteration is increased in order to lower the overall cost of search.

4.2 Ruffled by Ridges: Local Search on Ridge Functions

Local search, like the method of steepest-descent, is an appealing strategy partly because of its simplicity; the computational cost per iteration is minimal and almost no parameter tuning is required. In chapter 2, we stated that the behavior of local search is partly based on the precision. Precision controls the neighborhood size and the distance to the closest neighbor. Encoding can also have an affect. We have already discussed the relative merits of using the standard binary reflected Gray code. In this section, we explore how these two factors, encoding and precision, impact local search on ridge functions We also discuss how genetic algorithms, which often use a similar representation, may be affected by ridges.

4.2.1 Gray Encoding and Local Optima

Over the last few years experiments have shown that genetic algorithms are more sensitive to local optima induced by different bit representations than was previously believed. Until recently, much of this work has focused on how representations such as Gray codes destroy local optima [102]. This section focuses on when Gray codes *create* new optima: this happens only along *ridges*.

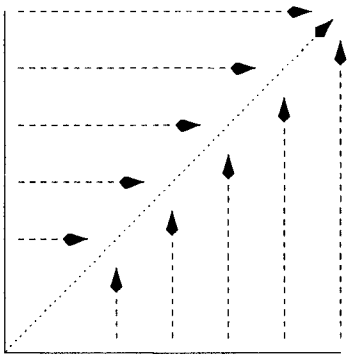


Figure 4.3: Local search moves only in the horizontal and vertical directions. It therefore "finds" the diagonal, but becomes stuck there. Every point on the diagonal is locally optimal. Local search is blind to the fact that there is gradient information moving along the diagonal.

Recall that local search search terminates at a *local optimum* when none of the points in its neighborhood improve upon the current best point when evaluated by some objective function. Gray codes are often used for bit representations, in both genetic algorithms and local search, because by definition, adjacent integers are adjacent neighbors. Whitley and Rana prove that if a one-dimensional function is unimodal under the natural encoding, it is also unimodal under Gray code [108]. But are there unimodal functions where the natural encoding is multimodal? If the function is not one-dimensional, the answer is yes. "False" local optima are induced on ridges.

A simplified *sharp ridge* problem appears in Figure 4.3. Changing one variable at a time will move local search to the diagonal. However, looking in either the x-dimension or the y-dimension, every point along the diagonal appears to be a local optimum. Local search fails on this problem, even though there is actually gradient information if one looks *along* the diagonal. However, this requires either changing both variables at once, or transforming the coordinate system of the search space so as to "expose" the gradient information.

This limitation is not unique to local search, and it is not absolute for genetic algorithms. Any method that searches 1-dimension at a time has the same limitation, including simple "line search" methods. A genetic algorithm is not absolutely trapped by ridges either. Early population sampling can allow the search to avoid being trapped on "ridges." But genetic algorithms can quickly lose diversity. If this happens, then the search must use mutation or otherwise random jumps to move along the ridge. For example, simple 1-point crossover inherits "fixed but mixed" parameters from

parents for the most part. That is, the inherited parameters come directly from the parents without changes except for one parameter that is broken by crossover. Uniform crossover would seem to have the ability to move along ridges: every bit is independently inherited from the two parent structures. But Syswerda [94] points out that when using uniform crossover, bits that are common between the two parents are inherited and all non-common bits are *randomly reset* because 0 or 1 is randomly inherited. So the ability of uniform crossover to move along ridges may be no better than that of random mutation.

One of the fundamental problems that is encountered when trying to compare direct search methods, local search methods, and even different evolutionary algorithms, is the representation and precision used for constructing the search space.

Genetic algorithms and local search [23] tend to use low precision bit encodings. Evolution strategies and direct search methods such as Nelder-Mead use high precision, real-valued representations. The search community has long struggled with the debate over which is better, bit representations or real-valued representations? Unfortunately, different experiments seem to support different conclusions, even when compared on the same test functions.

Recent work suggests that the choice of real-valued versus bit encodings may not be nearly as important as the level of precision. Precision can dramatically change the rate of convergence. One of the potential advantages of using bit encoding is that they can use lower precision for many applications and achieve faster convergence compared to real-valued encodings. This also makes comparing real-valued representations and bit encoded representations difficult. However, forcing the bit encodings and real-valued encoding to use 32 bit precision just to make them the same is probably not a reasonable solution: precision matters.

4.2.2 Precision Matters

Recall that the *local search* creates a neighborhood pattern by flipping one bit at a time. Since only a single bit changes, each neighbor will be aligned with one of the coordinate axis. Precision dictates how close local search looks for improving neighbors. If the ridge is poorly scaled and rotated, higher precision will be needed to find an improving move. A lower precision search will get stuck on the ridge, blindly assuming it has found a local optima because none of its neighbors

offer improvement.

Increasing the precision will generally decrease the number of false optima. This is because a higher precision search algorithm will be able to move further on the ridge and arrive at better solutions than an algorithm searching at a lower precision. Increasing precision, however, forces search to move very slowly through the landscape. This is partly because the neighborhood size is larger with higher precision. The more serious problem is that local search will take the *steepest* step, which, when the ridge axis is rotated away from the coordinate axis, will be one of the highest precision neighbors. This phenomena of taking extremely small steps is known as *creeping*.

Whitley *et al.* [104] first noticed that local search using 20-bits of precision required a surprising number of evaluations on both the Rana and Rosenbrock functions. Their examination of the problem revealed that local search was in fact creeping along the 45 degree ridges shown in Figure 4.1. In order to understand how creeping can impact local search, and also to explain why precision is so important on ridge functions, we executed local search on the 2D Rosenbrock and Rana test problems at 10 and 20 bits. Results are shown in table 4.1. The number of steps required to reach a local optimum jumps from about 200 steps under local search at 10 bits of precision to 200,000 or more steps at 20 bits of precision.

Function	Precision	Mean	Std	Steps	Std
Rosenbrock, 2D	10-bits	0.001	0.002	235	30
Rosenbrock, 2D	20-bits	4×10^{-7}	1×10^{-7}	2×10^{5}	4×10^{3}
Rana, 2D	10-bits	-501.9	06.0	225	22
Rana, 2D	20-bits	-503.0	04.8	3×10^{6}	8×10^{3}

Table 4.1: Results of steepest ascent bit climbing with 100 restarts at 10 and 20 bit resolution. Results are averaged over 30 runs. Mean is calculated using the best of the 100 restarts. Steps is the average number of steps needed to reach a local optimum.

As shown in Figure 4.1, both of these functions have ridge like properties. Search must move toward the upper corners to find the global optimum, as well as the better local optima. The behavior of local search on Rosenbrock's function is shown in Figure 4.4. The first south to north move occurs in a single step. The second west to east move also occurs in one step. The ridge is therefore encountered after only two moves. Because local search has sufficient precision and the ridge is

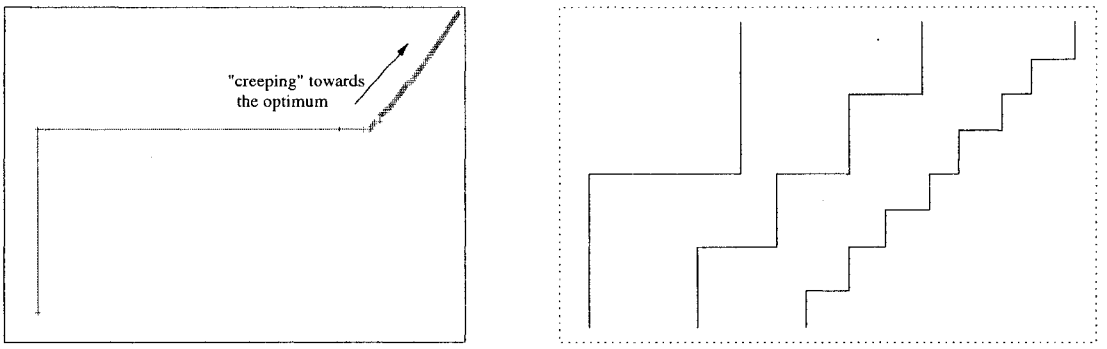


Figure 4.4: The left figure tracks the movement of an actual run of local search on Rosenbrock's function. After two moves the ridge is encountered. Then, local search begins to take small steps and creep toward the optimal solution. A higher precision search will be able to move through more narrow ridges. The problem is, adding 1 bit of precision in two dimensions doubles the number of steps local search will take. This is shown in the rightmost figure.

continuous and smooth (e.g. not a sharp ridge), local search is not completely blind and does not stop. Instead, the ridge becomes a "staircase". Local search makes the smallest move possible and therefore "creeps". The problem is exponentially worse at high precision because the steps of the staircases are exponentially smaller. In two dimensions, adding a single bit of precision doubles the number of steps local search will take.

Figure 4.5 also graphically explains this problem on a simple parabolic ridge. The leftmost graph shows that the low precision local search induces a false local optima on the ridge. The higher precision search is able to move farther along the ridge and finds a better solution, but incurs a much higher cost.

Genetic algorithms are very often used at 10 bits of precision. We have tested various genetic algorithms at 20 bits of precision across a number of the common benchmarks discussed in chapter 3 and found that they can be 10 to 100 times slower to converge using higher precision. This may in part be due to the failure of genetic algorithms to exploit ridges in the search space.

4.2.3 Rotating Search Coordinates

One way to deal with ridges in the search space is to search along the direction of the ridge. Recall that this idea is used in the conjugate-gradient method [90], and other direct search methods such as Powell's conjugate directions [79], and Brent's Praxis algorithm [19].

The ridge direction can be computed in two ways: either use a Principal Component Analysis

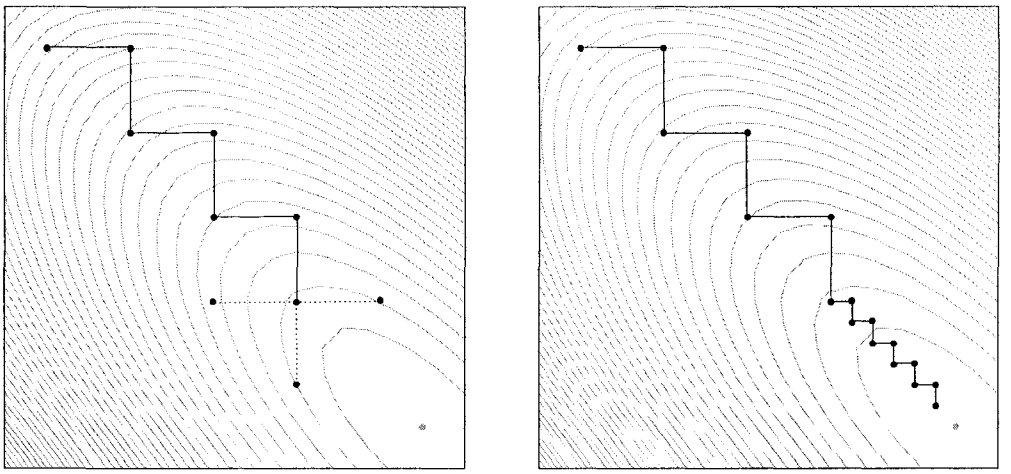


Figure 4.5: A low precision search induces local optima on a simple parabolic ridge because all the neighbors (dashed lines) have poorer evaluation. The higher precision search (rightmost) is able to make more progress, but at the expense of significantly more evaluations.

(PCA) on a sample of points or use the Gram-Schmidt (GS) orthogonalization algorithm to rotate the space based on a promising direction. Either way, we can change the search coordinates used by using a rotated representation.

Probably the more standard way of computing a rotation is to use Principal Component Analysis. Given a data set of sample points, an eigen decomposition is performed. The eigenvectors are represented by a rotation matrix $\mathbf{R}$. Let Λ be the diagonal eigenvalue matrix. Let $\mathbf{X}$ represent a matrix of data vectors. Using PCA we find $\mathbf{R}$ and Λ such that

$$\mathbf{R} \cdot \mathbf{X}\mathbf{X}^T = \Lambda\mathbf{R}$$

For a single search point represented by the vector $\mathbf{x}$ we compute $\mathbf{xR}$, which is the projection of the point $\mathbf{x}$ into the space defined by $\mathbf{R}$. The rotation matrix is orthonormal, so a simple correction is also needed to translate and re-center the rotation during search.

To find a structure such as a ridge, PCA can be used to sample locally and isolate a subset of the better sample points. For example, sample 20 points and then apply PCA analysis to the 10 best solutions. While this can give a correct rotation, the direction of maximal variance might be in the direction of the gradient if the samples are on a ridge, or the maximal variance may be orthogonal to the gradient if the sample is drawn from a sloping plateau.

Another approach is to use the Gram-Schmidt (GS) orthogonalization algorithm to rotate the

Functions	Search	Best	Mean	Std	Steps	Std	Evals	Std
Rosenbrock	LS	+4.5E-07	+5.4E-07	+1.2E-07	6,193	814	247,710	32,541
	PCA	+3.1E-10	+2.5E-07	+2.5E-07	138	58	7,603	3,189
Rana	LS	-510	-417	87	208	321	8,305	12,822
	PCA	-511	-480	24	23	6	1,262	341

Table 4.2: Results of local search and a rotated local search method. No restarts were used in these experiments. **Mean** is the mean best over 30 experiments, and best is the best of the 30 experiments.

space. Often the Gram-Schmidt algorithm is used to construct a representation that is orthogonal with respect to two points in the search space–such as the best two points seen so far. This is a heuristic way of determining a useful "rotation" for changing the problem representation.

In Table 4.2 local search (LS) with a Gray Code representation is compared with local search using PCA to rotate the search space. For the PCA, 15 points were sampled, with PCA applied to the best 8. The speed-up is *dramatic* using rotated representations.

We applied the local search algorithm with and without a PCA rotated representation to 5D and 10D versions of Rosenbrock's banana function and the Rana function. PCA was applied after every step, which adds to the number of evaluations. The number of steps taken during search is 5 to 10 times less under local search with PCA compared to the non-rotated representations on the 5D problems. The number of steps is 2 to 3 times less under local search with PCA compared to the non-rotated representations on the 10D problems. Using a rotated representation is more effective on Rosenbrock's function than on the Rana function. This may be because the ridge is the only feature that makes Rosenbrock's difficult, while Rana is also extremely multimodal. There is a clear advantage using PCA on the 5D problems; the advantage is less dramatic at 10D for Rana.

4.2.4 Discussion

The goal of this section is to highlight and explain the ridge problem as it pertains to local search and explore the use of rotated representations. Rotated representations could be used in conjunction with various types of evolutionary algorithms. Despite work in this direction, most of the evolutionary computation field has not looked seriously at rotated representations or the ridge problem.

We have already discussed at least one algorithm that already makes extensive use of rotated representations and has mechanisms to address some of the key questions that arise when using

Function	Search	Best	Mean	Std	Steps	Std	Evals	Std
Rosenbrock 5D	LS	2.4E-06	2.4E-06	8.1E-08	11,663	1,415	1,166,268	141,503
	PCA	5.3E-07	2.4E-06	1.3E-06	1,057	177	148,042	24,800
Rana 5D	LS	-386	-313	49	506	915	50,551	91,540
	PCA	-399	-310	42	112	167	15,662	23,318

Function	Search	Best	Mean	Std	Steps	Std	Evals	Std
Rosenbrock 10D	LS	5.9E-06	6.1E-06	1.3E-07	29,437	1,865	5,887,321	372,921
	PCA	3.8E-06	5.9E-06	2.2E-06	8,915	406	2,496,201	113,735
Rana 10D	LS	-427	-354	40	758	940	151,628	187,959
	PCA	-411	-308	47	525	632	146,917	176,958

Table 4.3: The results of applying local search with and without PCA rotated representations on 5D and 10D versions of the Rana and Rosenbrock functions. At 5D, PCA used the best half of 40 samples; at 10D PCA used the best half of 80 samples to compute rotations.

rotated representations: CMA-ES rotates and scales the mutation operators used by an evolution strategy. Other evolution strategies have tried this in the past also. In chapter 2 we briefly discussed *correlated mutations*, a concept that uses self-adaptation to evolve a set of rotation strategy parameters [8]. However, indirectly adapting rotation strategy parameters of the form used by this evolution strategies is too imprecise and impractical for large problems.

The key concern when computing rotations is: what kind of sample should be used? If a localized sample is taken and then the best points (e.g., the best λ individuals of the sample) are used to calculate the eigenvectors and eigenvalues, the direction of maximum variance can be in the direction of the gradient *or* it can be orthogonal to the gradient.

Recall that the Gram-Schmidt algorithm is often used to construct a representation that is orthogonal with respect to two points in the search space–such as the best two points seen so far. This kind of approach used less information, but emphasizes knowledge about gradient based on the most recent move or moves. This is a more localized and heuristic way of determining a useful "rotation" for changing the problem representation.

In effect, PCA exploits information about variance, whereas Gram-Schmidt uses path information. The path information acts as a kind of "momentum" term that keeps the search moving in it's current direction. Simple empirical experiments show that path information is most useful when

approaching a ridge, or when following a straight ridge. But path information is sometimes misleading, for example on a curved ridge. On a curved ridge, calculating the direction of maximum variance helps to track the changing gradient. These ideas are already exploited by the CMA-ES algorithm. CMA-ES uses path information, called "cumulation", which acts as a momentum term, and also uses information from the variance in the population, which is called the "rank-μ" update.

4.3 Searching for Balance: Self-adaptation and Ridges

Like local search, self-adaptation is a simple and often effective search method. It is invariant to rotations of the search space because it samples individuals in an unbiased direction. Therefore, unlike local search, it is not affected by the orientation of the ridge. At the same time, it has been empirically observed that the self-adaptive evolution strategy frequently generates a step-size that is sub-optimal on *ridge* functions. This causes self-adaptation to also *creep* or even to stall [44; 85; 43]. In this section we conjecture that the global step-size of a $(1, \lambda)$-ES will stabilize when the selection of σ is *unbiased* toward larger or smaller step-sizes. This occurs when the probability of selecting an individual with a smaller σ value is approximately equal to the probability of selecting an individual with a larger σ value.

We provide empirical evidence that shows when the ridge function is smooth (e.g. the parabolic ridge), self-adaptation will decrease its step-size until it is unbiased in selecting σ. On the sharp ridge, which is not continuously differentiable, the local topology in the neighborhood of the step-size does not change as σ decreases; this implies that for *any* σ value, there is a constant bias toward selecting individuals that have smaller step-sizes. Our explanation for this behavior supplements other justifications for the failure of self adaptation on the sharp ridge [85].

4.3.1 Self-adaptation and the General Ridge Function

Isotropic distributions are invariant to rotations of the search space. This means that the offspring are created in an unbiased *search direction*. The implication here is that the difference in scale between the parameters of the ridge, and not the ridge orientation, is the primary characteristic that can affect performance.

The generalized ridge function, as defined by Beyer [14], does not have an obtainable optimal

solution. Instead, this "treadmill" function is used to understand the steady-state behavior of search algorithms on the ridge. The goal is to maximize the following:

$$f(\vec{x}) = x_0 - d \cdot \left(\sum_{i=1}^{N} x_i^2 \right)^{\alpha/2}$$

where d is a *scaling* factor that determines how narrow and steep the ridge will be. The α constant determines the type of ridge; the parabolic ridge corresponds to an $\alpha = 2$ and the sharp ridge forms when $\alpha = 1$. Although the *ridge axis* (denoted x_0) is aligned with a coordinate axis, this will not skew our results because, as mentioned, isotropic distributions are unbiased to the orientation of the ridge axis.

The goal is to make progress in the x_0 direction while minimizing the distance to the ridge axis in all other dimensions [76]. This creates a problem for an isotropic distribution because offspring are sampled in an unbiased way. For example, there is no adaptive mechanism that allows self-adaptation with a global step-size to search for larger values in one direction while minimizing the other parameter values.

4.3.2 Related Work

The theoretical and empirical behavior of evolution strategies using *constant mutation strength* and a single global step-size is well documented for the parabolic ridge function [76; 74; 75]. Beyer extended this work by looking at the performance properties of the $(1, \lambda)$-ES on the general class of ridge functions [14]. Recently, Arnold and Beyer have studied CSA on the noisy parabolic ridge [5].

The behavior for an evolution strategy with a *fixed mutation strength* can be summarized using two important theoretical equations (see Beyer for details [14]). First, the distance of the best individual to the ridge axis tends to fluctuate within a predictable range, R, which is called the *stationary distance* [74]. This measurement is used in the second equation, the *progress rate* (φ), that predicts the average expected change in the direction of the ridge axis given a fixed σ value. Algorithms that *creep* on the ridge will tend to have lower progress rates.

Beyer points out that the results derived using a constant mutation strength can serve as a performance benchmark for *self-adaptation* [14]. If self-adaptation is working as expected, the steady-

state σ values should be close to the optimal predicted values using a constant step-size. Unfortunately, they are not.

Because an adaptive step-size is necessary in practice, it is important to understand how different adaptive strategies will perform. Oyman, Beyer, and Schwefel present conditions on the parabolic ridge where a $(1 + 10)$-ES *limps*, or *creeps* [74]. Several researchers have also noticed that non-elitist self-adaptation fails on ridge functions. Herdy showed that the self-adaptation with a single global step-size fails on the sharp ridge [44]. Salomon empirically showed that the $(1, \lambda)$-ES with a global step size failed on two instances of the *sharp ridge* [1] function [85]. Salomon found that the estimated *progress rate* is essentially negative for nearly all values $\sigma > 0$. Only when the population size was large enough, which Salomon found grew exponentially with problem dimension, did a small neighborhood of acceptable σ values yield positive progress.

Given the amount of attention devoted to evolution strategies on the ridge, it is surprising that no explanation has been put forth as to why self-adaptation tends to evolve less than optimal step-sizes on the general ridge function. In the conclusion of his paper, Beyer observes that self-adaptation appears to reward the short-term goals of reducing the *stationary distance* rather than the long-term goal of making *progress* along the ridge axis [14]. In the next section, we provide evidence that supports our conjecture as to why this occurs.

4.3.3 Why is self-adaptation sub-optimal?

On the ridge function there is a small "window" of improving search directions. This window gets smaller as either the dimensionality increases or the scaling factor, d, increases. This means that an unbiased isotropic distribution is more likely to sample the search space in a poor direction rather than an effective one. This creates a strong bias towards selecting smaller and smaller step-sizes.

In order to understand this bias, consider the three parabolic ridges in figure 4.6. The left ridge is sharp ($\alpha = 1$) whereas the other two ridges are parabolic ($\alpha = 2$). Two isotropic distributions based on σ are shown. One is slightly larger than σ and the other is slightly smaller. The base σ value in the first two graphs is 1 and decreases to $\sigma = 0.1$ for the right most graph. The "window"

[1] Salomon's test functions were similar to the general ridge function with $\alpha = 1$ and $\alpha = 0.5$.

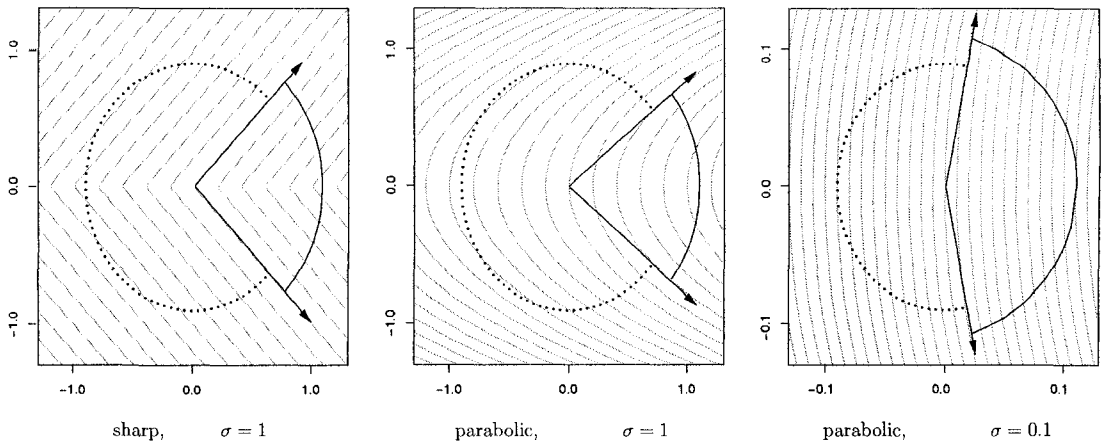


Figure 4.6: Contour plots of three ridge functions. The dotted arc in each plot corresponds to the region where the smaller step-size has greater fitness. The solid arc indicates the region where the larger step-size is best.

where the larger step size has greater fitness is less than the small step-size window in all cases. In other words, the smaller step-size has a larger region where its fitness is better than that of the larger step-size. This creates a bias towards selecting smaller σ values. Notice that on the sharp ridge, the "window" aligns perfectly with the contour lines; decreasing σ will not change the ridge bias. However, comparing the two right graphs shows that as σ decreases on the parabolic ridge, the neighborhood around the step-size becomes more linear and the bias towards smaller step-sizes is less pronounced. Given this inherent bias toward smaller step-sizes, we propose the following conjecture.

Conjecture 1. *The global step-size of a self-adaptive* $(1, \lambda)$*-ES will stabilize when the selection of* σ *is unbiased toward larger or smaller values. If the ridge bias cannot be removed, self-adaptation will continue to decrease* σ *by selecting smaller step-sizes.*

We found that the steady-state value of σ on the parabolic ridge occurs when the probability of selecting a small step-size is approximately equal to the probability of selecting a larger one. Self-adaptation decreases its step-size until the ridge bias diminishes. Intuitively, this makes sense. When σ is large, the *ridge bias* will drive σ towards a smaller value. If σ gets too small, the probability that a larger step-size will be selected increases. Self-adaptation tends to have a steady-state behavior that reflects this balance.

On the sharp ridge, however, the inherent bias cannot be removed. Although the sharp ridge is

continuous everywhere, its gradient is discontinuous along the ridge axis. This means that as the step-size decreases, due to the inherent ridge bias, the topology in the neighborhood around σ looks identical for all values. In other words, there does not exist a small enough step-size such that the bias toward selecting smaller σ values diminishes. This compliments Salomon's results [85]; in high dimensions, the population size needed to effectively sample the search space adequately is exponentially large. Without an adequate sample, an individual with a larger step-size will rarely search in the small window where the larger step-size is best.

The correlation between fitness and step-size is another way to understand *how* self-adaptation removes this bias. When the parent is located on the ridge axis and the current step-size is relatively large, the topology of the fitness function tends to look like a ridge. This means that there will be a positive correlation between fitness and step-size; smaller step-sizes will tend to produce the best individuals. On the parabolic ridge ($\alpha = 2$), as the step-size decreases, the topology around the parent appears more linear. This is because the function is continuous and differentiable everywhere. On linear fitness functions, fitness and step-size are uncorrelated [40]. That is, small step-size is just as likely to produce the best offspring as a large one.

In order to provide evidence for our conjecture, we would like to measure the probability that the step-size of the next generation parent is smaller than that of the current parent. If σ_g equals the step-size of generation g, then what we would like to measure is: $P(\sigma_{g+1} < \sigma_g)$. We denote this probability as ω. This will allow us to show that when σ reaches a steady-state, $\omega \approx 0.5$.

We can estimate ω by conducting a series of "single generation" experiments [2]. We do this by creating λ offspring based on a distribution defined by σ and the parent's location, $\vec{x}$. Then we evaluate the fitness of the offspring and ask the question: is the step-size of the best individual less than the value of σ? We repeat this 200 times and keep track of the number of times that a small step-size is successful. This is our estimate of ω, denoted $\hat{\omega}$.

We ran 100 trials of a $(1, 60)$-ES on both the parabolic and sharp ridge function. Each trial ran until either 1.) 1000 generations passed, which is ample to measure the stable behavior, or 2.) the step-size was below $1e - 10$, which is appropriate for measuring failure. After each generation,

[2] An idea used by Beyer [14] for a different purpose.

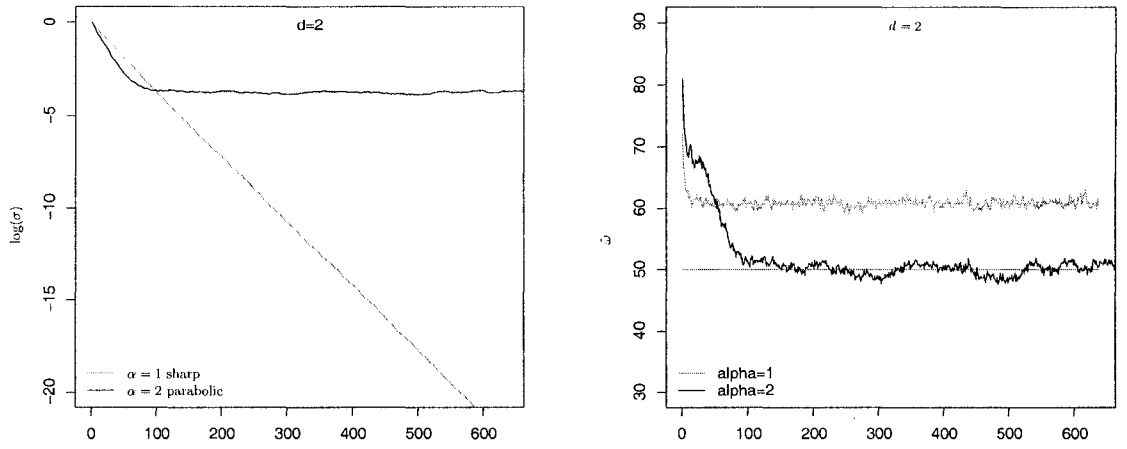


Figure 4.7: The left graph shows the convergence of $\log(\sigma)$ for $\lambda = 60$ on the sharp ($\alpha = 1$) and parabolic ($\alpha = 2$) ridge. The graph on the right shows $\hat{\omega}$. The x-axis is the generation number.

we estimated ω using the position and step-size of the current parent, as well as the corresponding population size, $\lambda = 60$. We tested several settings for the ridge scaling factor $d = \{1, 2, 5, 10\}$. Larger values of d create a steeper ridge function, which, given the same value for σ, increases the bias toward selecting smaller step-sizes. Figure 4.7 illustrates how self-adaptation behaves on a $N = 30$ dimensional ridge function where $d = 2$. The left graph shows the logarithm of σ for each generation. On the parabolic ridge, σ decreases until it reaches a steady-state value. Although this is sub-optimal, the strategy continues to make progress along the ridge axis. The step-size on the sharp ridge continues to decrease, never reaching a stable value. This is consistent with previously reported observations [44; 85]. The right graph shows the average $\hat{\omega}$ for each function. The value for $\hat{\omega}$ is approximately 50% on the parabolic ridge. The step-size decreases until the topology around its expected distance from the ridge is unbiased toward selecting small or large values of σ. On the other hand, as soon as the self-adaptation finds the sharp ridge, the value for $\hat{\omega}$ is constant and greater than 50%. This is because changing the step-size does not impact the bias toward selecting smaller σ values. Therefore, the step-size will continue to decrease.

Increasing the scaling factor d increases the rate at which the last $N - 1$ parameters decrease. This makes the difference between the rate of change for ridge axis (x_0) and all the other parameters more pronounced, and therefore, increases the *ridge bias*. There are two implications here. First, the steady-state σ values on the parabolic ridge will decrease as d increase. This is because it will take a smaller step-size to create an unbiased selection operator required for σ to stabilize. Second,

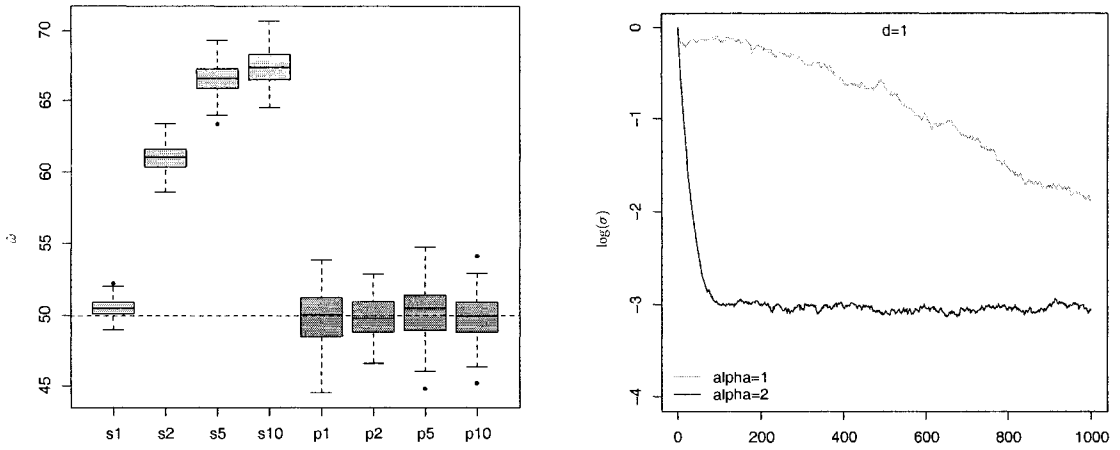


Figure 4.8: The graph on the left shows $\hat{\omega}$ for all values of the ridge scaling factor d. Each experiment is denoted with an s or p, depending on whether the ridge is sharp or parabolic, and the d value. For example, $s1$ refers to the sharp ridge with $d = 1$. The right graph shows the the convergence of $\log(\sigma)$ for $\lambda = 60$ on the sharp with $d = 1$. Although $\hat{\omega}$ appears to be close to 50 for the sharp ridge with $d = 1$ (s1), it is in fact significantly higher (using a t-test with a 0.95 confidence interval).

because the bias in the sharp ridge cannot be removed, an increased ridge bias will cause the step-size to decrease at a higher rate.

Figure 4.8 shows how the scaling factor d can affect self-adaptation on the sharp ridge. The right box plot indicates that increasing d creates a larger bias toward selecting smaller step-sizes for the sharp ridge. The parabolic ridge still reduces it step-size until $\hat{\omega} \approx 50$, but this requires a smaller σ for larger values of d (which is not shown here). Although the $\hat{\omega}$ value for the sharp ridge with $d = 1$ appears to be close to 50, it is actually significantly different. Even this small bias will cause instability in the step-size. The right graph of figure 4.8 shows the convergence behavior for σ on the ridge functions with $d = 1$. The parabolic ridge balances when $\hat{\omega} \approx 50$. On the sharp ridge, a slight bias in selection ($\hat{\omega} > 50$) causes the step-size to decrease slowly. Given 1000 generations, self-adaptation makes greater progress on the unscaled ($d = 1$) sharp ridge than it does on the unscaled parabolic ridge. This is not surprising because the parabolic ridge has a much larger rate of decrease to the ridge axis than the sharp ridge, which decreases linearly.

4.3.4 Discussion

We have provided evidence to support our conjecture that self-adaptation will continue to decrease its step-size on the parabolic ridge function until it removes the bias toward selecting smaller σ values. On the sharp ridge function, this inherent bias cannot be removed, and σ will decrease on

average. This explains why the performance of self-adaptation is poor on ridge functions.

The shortcomings of *correlated mutations* and the lack of robustness that frequently occurs when adapting *individual step-sizes* leave an impression that the *only* safe way to use self-adaptation is with a single strategy parameter. And this strategy can also fail on the *sharp ridge*, a relatively harmless looking unimodal surface. To confuse the issue even more, when self-adaptation does not fail, it usually adapts step-sizes that are too small. All of these arguments seem to indicate that the self-adaptive assumption – highly fit individuals will also have useful strategy parameters – is incorrect.

On the other hand, the selected individuals *will tend to have* distributions that are more likely to explore better regions of the search space on the ridge function. A large step-size is less likely to "explore" better regions of the search space because of the inherent bias that comes with ridge functions.

4.4 Summary

We have explored why ridges are difficult for local search and the self-adaptive evolution strategy. For local search, changing only one variable at a time creates a fixed neighborhood search direction. Ridges that do not align with this coordinate pattern will cause local search to creep or even stall if the ridge is sharp. Similar problems exist for other pattern search methods such as GPS. Changing the search direction for these algorithms can increase their efficiency, but getting the right direction adds complexity and is difficult as the number of parameters increase.

The self-adaptive evolution strategy decreases its step-size until it finds a balance between accepting larger and smaller values of σ. Like local search, this makes self-adaptation also creep on smooth ridges and fail on sharp ridges. CMA-ES overcomes this problem by altering its search distribution based on path information and the best points of the current sample.

One common theme in optimization that continues to emerge across different disciplines is this: unless an algorithm specifically exploits covariance information, it is unlikely that its heuristics, either direct or using the gradient, will perform well on ridge problems. Furthermore, it seems that algorithms that do address the ridge problem must incur additional complexity in order to gain overall efficiency. That is, finding the ridge direction is difficult and appears to come with

a computational penalty. Simple methods, like steepest-descent, local search, and self-adaptation, are all less efficient, or even fail, on ridge surfaces, while more complex algorithms, such as a quasi-Newton method or CMA-ES, are overall more efficient and effective.

In terms of difficulty, the ridges used in this chapter are relatively easy. For example, the condition number for Rosenbrock's function is $\kappa \approx 2.5e3$. This number also does not increase with dimensionality. A function is generally considered ill-conditioned when it has a value $\kappa > 10e6$. This means that, although local search and self-adaptation were able to creep along the ridges in this chapter, numerical precision may cause these methods to fail on problems with more dramatic ridge features. The evolutionary algorithm community should pay more attention to ridge features.

Chapter 5

Applying Search Algorithms to the Temperature Retrieval Problem

Our exploration of the ridge problem in chapter 4 was partly motivated by our work with a real-world inverse problem from the atmospheric sciences, the *temperature retrieval* problem. This problem seems to be a good application for heuristic search because, although traditional optimization methods can be used, the gradient is sometimes not available or is extremely costly to compute. This can make the search times slow (e.g. hours). However, we have found that some evolutionary algorithms do not work well on this problem. We now know that there are *ridges* in these search spaces and that, as we discussed in chapter 4, they can induce "false" optima for algorithms that use bit-encoding, and are equally as challenging for search strategies, like self-adaption, that sample the search space in an unbiased way.

The *temperature retrieval* problem is a *forward model* that relates vertical temperature profiles to observed measurements. Researchers working in the atmospheric sciences actually need the inverse: given the observed data, what temperature profile produced those observations? Every set of observations that is collected results in a new temperature inversion problem that must be solved. These temperature profiles, which are used in global atmospheric circulation and weather prediction models, must be found efficiently in order to process a days' collection of data in a day.

We formally present results for a variety of search methods: CHC, CMA-ES, the self-adaptive evolution strategy, which we again denote SA-ES, and local search. A number of algorithms have been been applied to the temperature retrieval problem on a more limited basis, including Population-Based Incremental Learning, or PBIL [11] and Differential Evolution [93]. With the

exception of CMA-ES, all of these algorithms fail in similar ways. Even the useful solutions found by CMA-ES take too long to be of practical use. We show that the temperature retrieval problem is difficult for these heuristic search methods because there are ridges in the search space.

The inefficiency—and even failure—of these methods naturally raises a practical question: what is the best way to find useful temperature profiles efficiently? It is usually the case that the more knowledge an algorithm has about a particular problem, the more efficiently it will be able to find better solutions. In light of this observation, we investigate how to exploit two knowledge-specific features of the temperature problem.

First, since this model measures some physical phenomena, the expectation is that spatially close object parameters will tend to have similar temperature values. Salomon has proposed an *optimize and refine* algorithm that exploits the smoothness of temperature profiles and produces useful results efficiently when compared to the standard evolutionary algorithms we tested [84]. We show that even with this added problem-specific knowledge, not all the solutions found by this method are acceptable. We respond to this by presenting a new algorithm called "Tube Search". Our algorithm uses smoothness constraints to avoid ridge problems and quickly produces good approximate solutions.

Second, the way the objective function calculates the quality of a particular solution is not arbitrary; it is a nonlinear least-squares problem. Recall from chapter 2 that the structure of this problem is exploited by the Levenberg-Marquardt algorithm. We show that this gradient-based strategy is extremely fast on this problem.

This chapter is built around the temperature retrieval problem in the following way. The next section describes the temperature retrieval problem in greater detail. Then, we discuss the results of several search algorithms that we have applied to the temperature problem. We show that the temperature problem is difficult for search because ridges exist in its fitness landscape. Then we explore two algorithms that exploit the physical continuity of the temperature problem: Salomon's optimize and refine method and our new algorithm, tube search. Finally, we show that exploiting objective function structure and using gradient information is incredibly fast on this problem.

5.1 The Temperature Retrieval Problem

Researchers working in the atmospheric sciences have created a *forward model* that relates vertical temperature profiles to observed measurements. The forward model, as described in this paper, generates 2000 radiance measurements (observations) given a 43 dimensional temperature profile. The k^{th} parameter in the profile is the estimated temperature at an altitude of approximately k kilometers in the atmosphere (in reality, the spacing is somewhat greater at higher altitudes). We actually want to solve the inverse problem: given a set of observations, what is the corresponding temperature profile? In practice, radiance measurements from a constellation of satellites are used in an inverse radiative transfer model. Examples of extant observing systems are: Operational Vertical Sounder (OVS), the Special Sensor Microwave Imager (SSM/I), and the Advanced Microwave Sounder Unit (AMSU). The inverse solution must be accurate and *fast*; measurements are often collected at a high spatial resolution from satellites whose orbital period is about 90 minutes (i.e. moving at about 8 km /sec).

The forward model is the simplified form of the equation of radiative transfer that does not account for the presence of clouds. The equation of transfer is solved for the radiances at different wavelengths observed at the top of the atmosphere. This radiative transfer model is "plane parallel" (e.g. one with no horizontal variations in its properties). The radiances are calculated at a certain viewing angle θ as:

$$I_{(\tau,\mu)} = B_\nu(T_s)e^{-\tau_s/\mu} + \int_0^\nu B_\nu(T)e^{-\tau/\mu}\mu^{-1}d\tau$$

where $I_{(\tau,\mu)}$ = radiance

μ = $cos(\theta)$

τ = optical depth

s = surface

$B_\nu(T)$ = Planck radiance for temperature T.

An analytical inversion of this model is impossible. Alternatively, the inverse temperature model can be formulated as an optimization problem, where the target temperature profile is the global

optimum of the search space. Specifically, the objective function becomes a nonlinear least-squares problem where the fitness is the sum-squared-error between the the observable measurements, Y_i, and the output of the forward model at a given temperature in the search space, $F(\vec{x})_i$.

$$f(\vec{x}) = \frac{1}{2}\sum_{i=1}^{M}(F(\vec{x})_i - Y_i)^2$$

First order derivatives can be calculated analytically for the temperature retrieval problem [30]. In the more complex and realistic models, the analytical calculation of derivatives is impossible. Currently, success has been achieved using the Gauss-Newton iterative method and a good starting profile. Although the solution can be accurate, achieving a quadratic convergence rate for solutions near the optimum is highly dependent on a good apriori guess of the temperature profile. Later in this chapter, we show that the Levenberg-Marquardt algorithm works well from any random starting point.

5.2 Empirical results

We used the well-known McClatchey *tropical* temperature profile [64] as our empirical search objective. Experiments performed on other McClatchey profiles indicate that this target profile is representative of the problem difficulty.

In this section, we compare CHC, CMA-ES, the self-adaptive evolution strategy (SA-ES), and local search (LS) on the temperature problem. As mentioned, the precision for CHC is typically set between 10- and 20-bits. A higher precision search will tend to restart less often because each individuals representation considers more information, and therefore, will maintain diversity longer. In our experiments, we used 20-bits.

Precision was also the only parameter we considered adjusting for local search. We have seen that higher precision will converge more slowly because the neighborhood pattern is larger and the potential to *creep* is greater. This becomes a practical problem in high dimensions. For example, a 20-bit LS on the temperature problem requires 860 neighborhood evaluations in order to take a single step. In order to make LS more efficient, we used 10-bits of precision, which cut the neighborhood size in half. We know from the previous chapter that this precision choice will likely induce more local optima on ridges than a higher precision search.

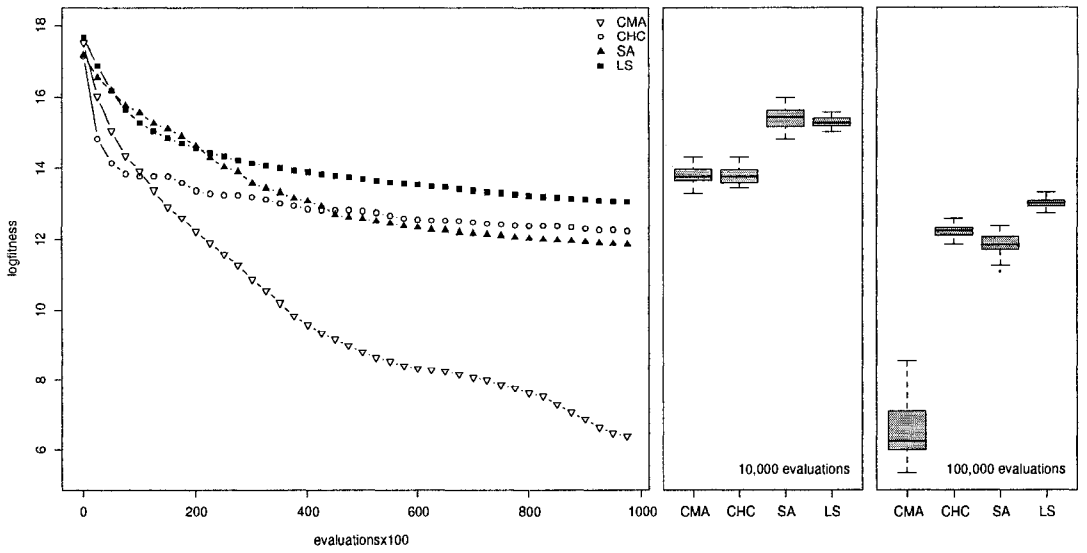


Figure 5.1: The convergence of each algorithm, based on log(fitness), with respect to the number of evaluations. The right two graphs are box plots of log(fitness) for each algorithm after 10,000 and 100,000 evaluations. Only CMA-ES is able to find acceptable solutions to this problem. Unfortunately, using 100,000 evaluations is impractical for real-time performance.

We used a population size of $\lambda = 100$ for both CMA-ES and SA-ES. The initial step-size was set to $\sigma = 30$, which is 25% of the domain. Our preliminary results indicated that the larger recommended step-size values did not yield better results. We also found that smaller population sizes were less effective. CMA-ES used the default parent number and weighted intermediate recombination ($\rho = \mu$). We used a single parent ($\mu = 1$) for SA-ES.

Each algorithm ran for 30 trials and each trial used exactly 100,000 evaluations. Random *restarts* were used to ensure that every trial used all of its allocated evaluations. While 100,000 evaluations may seems small, or arbitrary, for a problem of this dimension, we actually need to reduce the number of evaluations significantly to achieve real-time performance.

The median convergence for each algorithm is plotted in Figure 5.1. The box plots on the right show the fitness of the solutions found by each algorithm after 10,000 and 100,000 evaluations. The most striking observation in Figure 5.1 is that CMA-ES is dramatically more effective on this problem than the other algorithms tested. And, although each search strategy continues to improve with an increased number of evaluations, the rate at which CMA-ES is improving is significant. What is unclear from this data is exactly *how* useful any of these solutions are.

The five median solutions (out of 30) found by each algorithm after 100,000 evaluations are

displayed in Figure 5.2 along with the McClatchey *tropical* profile. These solutions are indicative of the expected performance. The solid black line is the target profile; the gray lines are the five median solutions. From a practical point of view, CMA-ES is the only algorithm capable of finding solutions that are a reasonable fit to the desired temperature profile. All the other median solutions zig-zag the target profile too much to be of any practical use. Unfortunately, even the useful solutions obtained by CMA-ES take an unrealistic number of evaluations.

We also observe from Figure 5.2 that the solutions found by each search algorithm fit the lower parameters of the profile better than they fit the upper dimensions. This is most pronounced in the SA-ES results, but still noticeable in the other algorithms. For example, the median solutions of SA-ES fit the profile well between parameters 2 through 20, but zig-zag the profile elsewhere. Although the solutions found by CMA-ES are close to the target, we found that it assigns the correct values to the lower dimensions first, and then continues to search for more effective values in the upper parameters second.

5.3 Problem difficulty

Our results show that CMA-ES has a clear advantage on the temperature problem. One interpretation of this performance gap is that the heuristics employed by CMA-ES interact well with the surface features of the this problem. In chapter 3, we observed that CMA-ES was the only algorithm that performed well on ridge features and is the only method that presented here that has heuristic mechanisms in place that allow it to travel along ridges in the search space. The dominating performance of CMA-ES here is a good indication that there are ridges in the search space of the temperature problem. The question we explore in this section is: what are the problem features that make the temperature retrieval problem difficult for search?

5.3.1 Separability, Ridges, and Bias

We know that the problem is highly *non-separable*. There exists a strong *non-linear* interaction between adjacent parameters. For example, finding the correct temperature value for parameter k will imply changing the values of the parameters that are in the range $k-i$ to $k+i$. The neighborhood size, i, of this local interaction is unknown. The closest parameters are likely to have the strongest

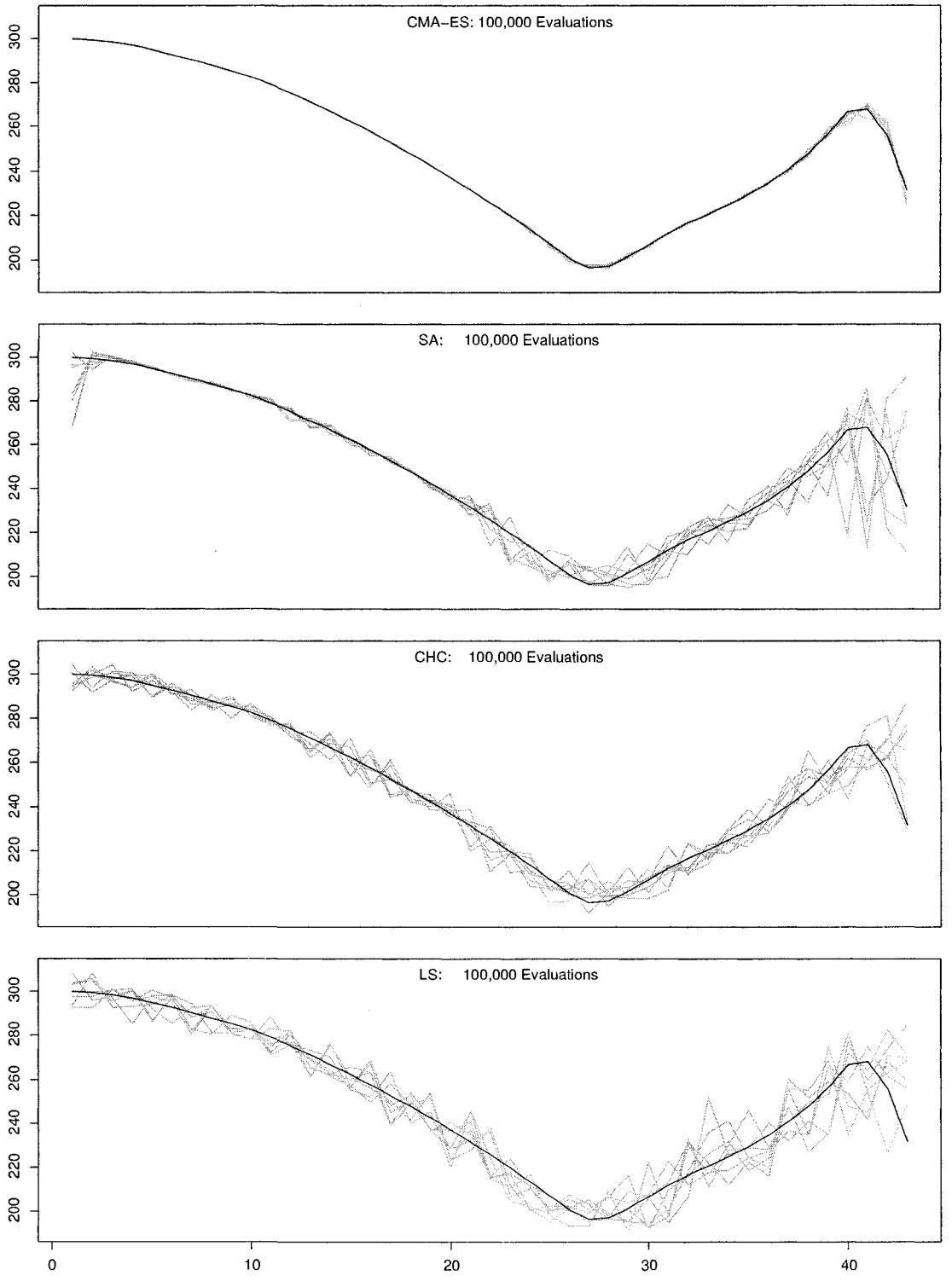


Figure 5.2: The five median solutions found by each algorithm. The solid black line is the target profile. The gray lines are the actual solutions. Notice that all algorithms pay more attention to the lower dimensions of the profile. These are the parameters that offer the largest rate of descent. This is most noticeable in Self-adaptation (SA).

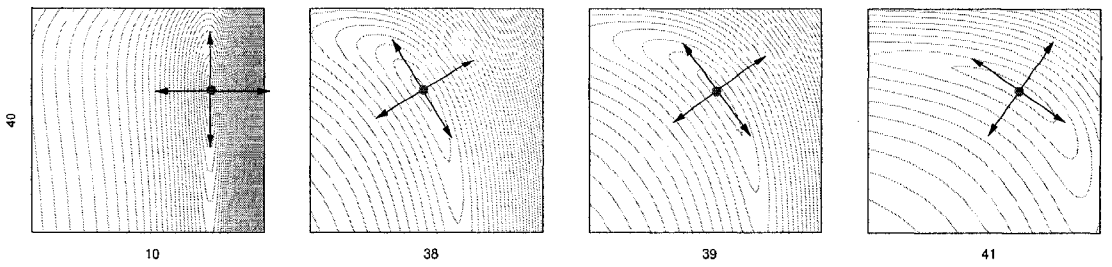


Figure 5.3: Two-dimensional surfaces created by varying the two dimensions of the optimal solution. The axis in each case is aligned with the ridge and shows the degree to which the parameters interact near the optimal solution. The surface on the left is *poorly-scaled* and the interaction between the variables is minimal. The surfaces to the right are more evenly scaled, but there is also a strong interaction between the variables.

interactions while more distant dimensions will exhibit a weaker correlation.

As expected, we have also noticed that there are *ridges* in the search space. Figure 5.3 shows several representative 2D slices of the temperature problem's search space. Each surface was created by changing two parameters of the optimal solution. For example, to create the left-most surface, we varied parameters 10 and 40. The resulting surface shows that there is very little interaction between these two variables, but that there exists dramatic difference in scale; the rate of change in parameter 10 dominates the landscape. The other three contour plots in Figure 5.3 show the interaction of parameter 40 with parameters 38, 39, and 41. Although the difference in scale is less pronounced in these other slices, there exists a much stronger interaction between the variables. This is because each of these surfaces involve parameters that are in the neighborhood of parameter 40 (e.g. 38, 39, and 41).

We believe that search algorithms pay greater attention to the lower dimensions of the temperature problem because these dimensions offer the greatest rate of decrease in error. Starting from the globally optimal solution, we varied each parameter by $\pm$ 2 units (degrees Kelvin). Every move increases the objective error, which is zero when no change is applied. The average change per dimension, denoted $\hat{b}$, is a rough estimate of the expected change in the objective function associated with each object parameter near the optimal solution.

Figure 5.4 shows the estimated bias $\hat{b}$ for the temperature retrieval problem. Notice that the parameters offering the greatest opportunity to reduce the error will correspond to the largest values of the estimated bias $\hat{b}$. This bias causes the search algorithms to fit the "steeper" dimensions of

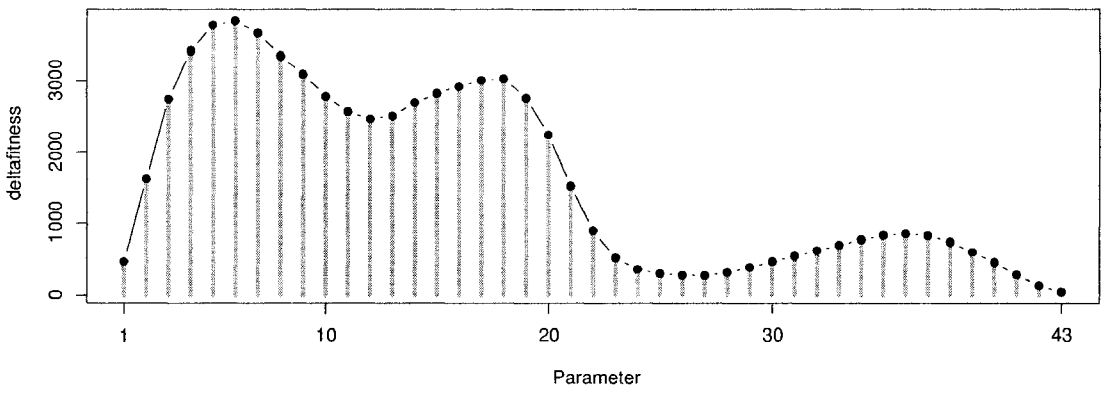


Figure 5.4: An estimation of the parameters that offer the greatest influence on the fitness function. The average bias per dimension is lower for the parameters upper dimensions of the problem.

the profile first – and to assign less precise values to the other parameters. Comparing the profile in Figure 5.4 to the results displayed in Figure 5.2 it becomes more clear that each algorithm favors the parameters that have the greatest bias.

5.3.2 Computing the Condition Number

In the previous chapter, we mentioned that the condition number of the Hessian is representative of how poorly-scaled a surface is. For the Temperature problem, computing the Hessian analytically may be possible, but would be extremely challenging. Furthermore, the Hessian is almost never used directly in search; the most efficient nonlinear optimization methods use first-order information to approximate the Hessian. Yet it would still be informative to have an idea of how difficult the ridge features in the temperature problem are.

With very little effort, we can compute the exact Hessian around the optimal solution using only first-order information. This is a well-known intrinsic property of any least-squares objective function. Gradient-based optimization algorithms like the Gauss-Newton and Levenberg-Marquardt use this property as part of their heuristic for estimating the Hessian. Recall that the objective function is of the form:

$$\begin{aligned} \text{minimize} \quad f(x) &= \frac{1}{2}\sum_{i=1}^{m} r_i(x)^2 \\ &= \frac{1}{2} R(x)^T R(x) \end{aligned}$$

$$\text{where} \quad R(x) = F(\vec{x}) - Y$$

The gradient is:

$$\begin{aligned}\nabla f(x)_j &= \frac{\partial f(x)}{\partial x_j} \\ &= \sum_{i=1}^{m} r_i(x)\frac{\partial r_i(x)}{\partial x_j}\end{aligned}$$

We can define the Jacobian as:

$$J(x)_{i,j} = \frac{\partial r_i(x)}{\partial x_j}$$

Substituting the Jacobian notation into the above gradient and writing this in matrix form we have:

$$\nabla f(x) = J(x)^T R(x)$$

The Hessian is:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_i}\left(\frac{\partial f(x)}{\partial x_j}\right) \\ &= \frac{\partial}{\partial x_i}\left(\sum_{i=1}^{m} r_i(x)\frac{\partial r_i(x)}{\partial x_j}\right) \\ &= \sum_{i=1}^{m}\left(\frac{\partial r_i(x)}{\partial x_j}\frac{\partial r_i(x)}{\partial x_j} + r_i(x)\frac{\partial^2 r_i(x)}{\partial x_i \partial x_j}\right)\end{aligned}$$

If we denote

$$S(x)_{i,j} = r_i(x)\frac{\partial^2 r_i(x)}{\partial x_i \partial x_j}$$

then the Hessian in matrix form becomes:

$$\nabla^2 f(x) = J(x)^T J(x) + S(x)$$

The important point here is that when the candidate solution $\vec{x}$ is at the optimal temperature, the second term in the Hessian, $S(x)$, goes to zero because the residual term $r_i(x)$ is zero. This means that we have the exact Hessian around the optimal temperature solution by using only the Jacobian, $J(x)$, which is comprised of first-order information only. When we do this we find that the condition number of the temperature problem at the optimal solution is $\kappa \approx 5.8e10$, an indication that near the optimal solution, this problem is highly ill-conditioned. Recall from chapter 4 that the condition number of Rosenbrock's test function was only $\kappa \approx 2.5e3$.

5.4 Directly Exploiting Problem-Specific Information

So far we have observed that the only *general-purpose* algorithm that found acceptable solutions to the temperature problem is CMA-ES. We use the term general-purpose here because there is nothing about CMA-ES—or any of the other algorithms tested so far—that exploits problem-specific apriori knowledge about the temperature problem. But the more knowledge an algorithm has about a particular problem, either *directly*, based on problem-specific information, or *indirectly*, based on its relative bias, the more efficiently it will find better solutions. We ask the question: can we directly exploit some characteristics of this problem to increase efficiency?

In this section, we exploit two specific features of the temperature problem. First, the temperature retrieval problem is one instance of an application that measures some physical phenomena. The expectation is that spatially close object parameters will tend to have similar values. This creates a "smooth" expected profile through the parameter values being optimized. This characteristic can be exploited by imposing a smoothness constraint on the parameter values of the objective function.

Second, the objective function for a nonlinear least-squares problem is not arbitrary. There is structure that exists in this function that is specific to a least-squares problem and can be exploited. In the previous section, we demonstrated that the exact Hessian can be computed using only first-order information when the trial vector is optimal. We discussed how the Gauss-Newton method uses only the Jacobian to approximate the actual Hessian even when the trial solution is not optimal. The Levenberg-Marquardt also makes this assumption, but is more robust than the Gauss-Newton method. We show that this algorithm works extremely well for the temperature problem.

5.4.1 Physical Continuity

Salomon [84] found that self-adaptive evolution strategies produced unsatisfactory results when the desired solution displayed physical smoothness. Salomon created an artificial test function with physical continuity to demonstrate that a (1,6) self-adaptive evolution strategy (with a single global step-size) often fails to converge to the optimal solution. The test problem used a simple convex function (i.e. $f(x) = 1 - x^2$) as the target profile. The fitness of an individual was an approximation of the area between the current solution and the target profile. Figure 5.5 shows this test function.

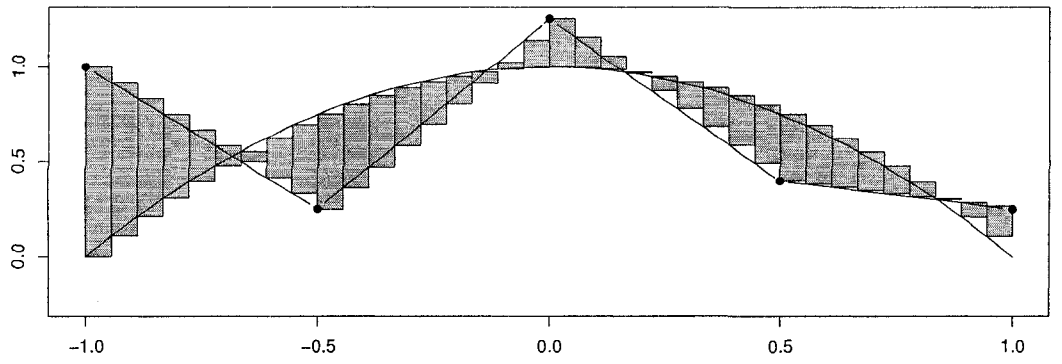


Figure 5.5: Salomon's test function: Fitness is calculated as the sum of the gray rectangles, which are an approximation of the area between the current solution (zig-zag line) and the target profile.

This is consistent with our results on the temperature problem. In fact, Salomon's solutions often "zig-zag" the target profile producing results similar to the self-adaptation results in figure 5.2.

We replicated Salomon's results on 43 dimension version of this artificial test problem. Figure 5.6 shows a typical solution. Salomon attributes the evolution strategies poor performance to a failure in step-size; by the time that most of the parameters have converged, the global step-size is small and it is increasingly unlikely that a parameter far from the target profile will progress towards to optimal solution. That is, a small global step-size adapts because the best individuals of the population are likely to be close to the current solution, which in turn is near the target profile. With a small global step-size, the outlier points are unlikely to converge on the target profile. This leaves the undesirable "peaks" or "zig-zags" in the solution.

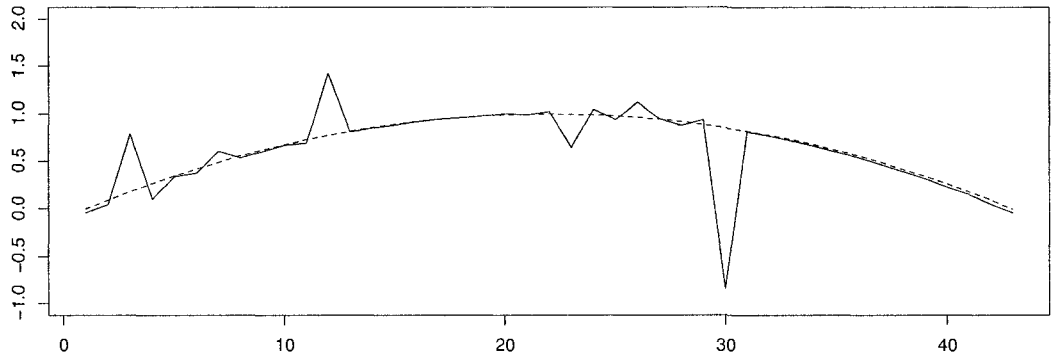


Figure 5.6: A typical solution found by SA-ES on Salomon's artificial test function.

We also tested CMA-ES, CHC, and local search on this problem. We found that the self-adaptive evolution strategy was the *only* algorithm that failed to find the optimal solution. There is nothing inherently difficult with problems that have optimal solutions exhibiting physical continuity.

Salomon's objective function is difficult for self-adaptation because the discontinuous nature or the objective function creates *sharp ridges* in the fitness landscape. In chapter 4, we showed that on sharp ridges, self-adaptation is unable to remove the bias between small and large step-sizes, and therefore, continues to decrease its step-size until it fails. However, this simple problem does not have the same degree of non-linearity and scale that makes the temperature problem difficult for the remaining algorithms. Local search is actually successful on this problem because the sharp ridges are closely aligned with the coordinate axis.

Optimize and Refine

Salomon suggests an *optimize and refine* evolution strategy as a means of avoiding the problematic "peaks" shown in Figure 5.6. The *optimize and refine* technique was inspired by manufacturing methods: many products start with a rough approximation that is refined to be more smooth. The smooth target profile of the temperature retrieval problem may be tackled in the same way. The procedure starts by approximating the target with a linear fit. The endpoints, x_1 and x_{43}, are searched for the position where linear interpolation minimizes the objective error. Refinement reduces the regions by half, and the solution becomes a piecewise linear approximation. For example, the next iteration would increase the dimensionality from two to three by adding the point x_{20}. This two piece linear approximation is optimized before more points are added in the next refinement phase.

This method is efficient in several ways. First, a close approximation to the target is found by searching small landscapes. In the temperature retrieval problem, a linear approximation reduces the dimensionality of the search space from 43 to only two. This gives higher dimensional searches a good place to start. Second, it forces a smoothness constraint on the problem. Neighboring points in the domain are forced to be relatively close in the range.

Salomon used a (1,6) evolution strategy in his optimize and refine method. We implemented a simple *binary search* to locate the minimum at each inflection point of the piecewise linear solution. The search started at the endpoints, x_1 and x_{43}. The *binary search* moved to the optimum in each dimension for several iterations until no improvement could be found. Then, the point x_{20} was added to break the linear region in two, and the optimize procedure was repeated, this time with three points instead of two. At each step, the regions, defined by the current set of points, were cut

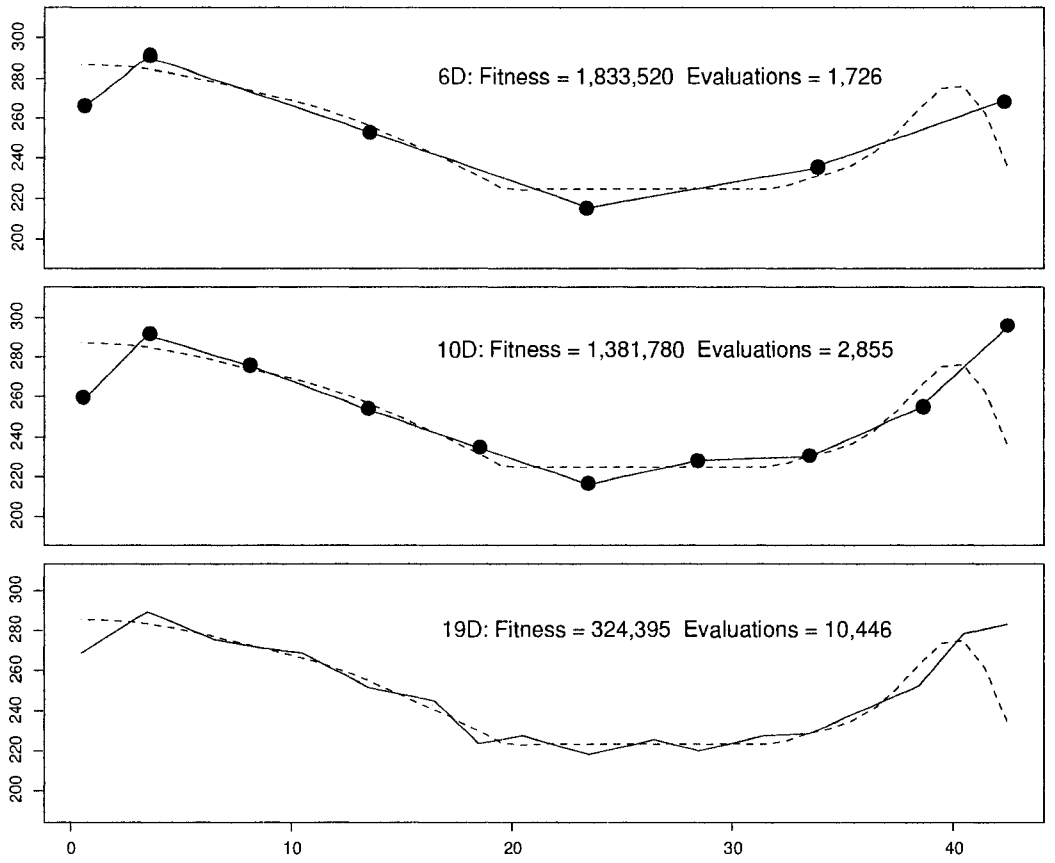


Figure 5.7: Optimize and Refine. A fully convergent *subarctic summer* solution required an average of 10,446 evaluations. Although this solution fits the profile better than previous methods, it still zig-zags the target solution.

in half after they had been fully optimized. Figure 5.7 shows this procedure for the McClatchey *subarctic summer* profile. Although this method shows promise, it is able to fit some data examples better than others. Sometimes the solution still "zig-zags" the target. This method also struggles to fit the ends of the profile.

A New Algorithm: Tube Search

We know that the target temperature profile we are trying to retrieve is relatively smooth, a constraint that is exploited by the *optimize and refine* algorithm. In response to this, we implemented a new algorithm called *tube search.*

Like optimize and refine, tube search starts with a linear fit. This provides a consistent starting point that is smooth, a quality we hope to retain throughout the search. Once the linear fit has been determined, tube search begins. A fixed step is taken on either side of the linear fit – in effect

defining a tube about that solution – and the change in evaluation is recorded and stored in a vector. Some moves will offer improvement, while others will not. Once improving moves have been determined, a step of the same magnitude is taken in each improving dimension simultaneously. A three-parameter moving average is run on the solution every five iterations to maintain a smoothness. Each parameter, except the first and last two end points, is replaced by the average of itself and its two neighbors.

$$\text{solution}[i] = \frac{\text{solution}[i-1] + \text{solution}[i] + \text{solution}[i+1]}{3}$$

Figure 5.8 graphically explains the tube search and shows the final solution generated by searching the temperature problem. Note that 43*2 evaluations are needed to evaluate the moves defined by the tube. Given the small number of moves used by the tube search, the total number of evaluations is less than half of that used by the the optimize and refine algorithm.

The error values associated with the move forming the *tube* around the current best solution will drive the search toward better points while maintaining smoothness. Because all of the parameters change at once, tube search is not a simple coordinate search scheme. Additionally, when each step is taken the magnitude of the step is the same independent of the magnitude of the error. In this way, tube search ignores the bias in the evaluation function. Higher dimension parameters can change just as much as lower dimension parameters, even when they have a smaller contribution to the error.

Tube search works surprisingly well on all temperature profiles we have optimized. Oddly enough, the errors associated with the tube search solutions are not particularly low: the errors are generally much lower for *optimize and refine*. Even CHC achieves lower errors. However, if we compute a sum-squared error (SSE) between the actual target temperature (which we don't have in the general case) and the tube search solution, the fit between the tube search solutions and the actual profile is better, on average, than is achieved with other methods. A smooth solution closer to the actual profile is more useful than a jagged solution with a lower error. Table 2 shows the *optimize and refine* method compared to the *tube* search method for all the McClatchey profiles we tested. The better objective fitness achieved in the *optimize and refine* algorithm does not imply a closer fit to the target solution. This may be because the other methods are more affected by bias in the evaluation function.

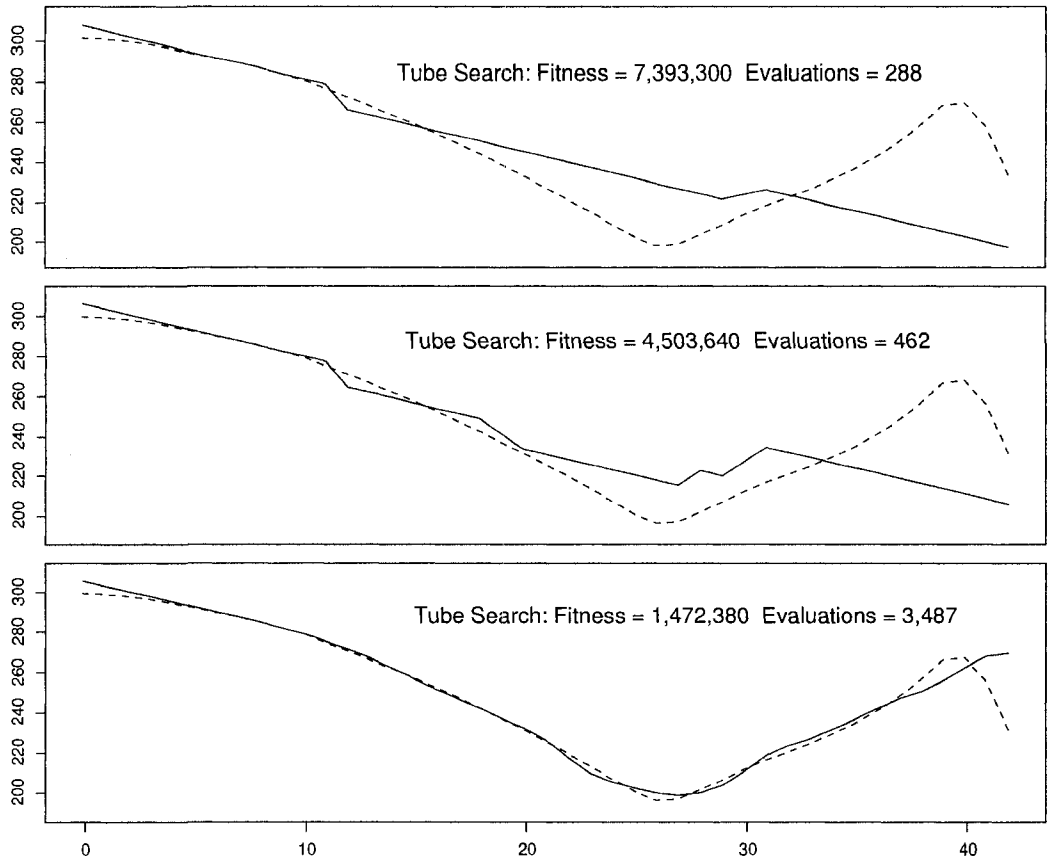


Figure 5.8: Tube Search: The top two graphs show select iterations of the tube search. The bottom graph shows the final solution after 3,487 evaluations. Of all the profiles tested, this was the worst fit. The step distance for each parameter is exactly the same, so bias has no impact on tube search.

Tube search is also much faster than the other methods using fewer than 3,612 evaluations on all data sets. This is still not fast enough to allow for real-time evaluation. However, tube search has another attractive feature. Each of the 86 evaluations required to evaluate the moves defined by the tube around the current best solution are independent and can be done in parallel. This would allow us to use parallelism to speed up Tube Search by a factor of 86. Parallel tube search could obtain a solution in the amount of time taken to do $3,612/86 = 42$ sequence evaluations. This is a major advantage given the goal of doing real-time temperature retrieval. For perspective, a parallel version of CMA-ES would requires roughly 1000 steps to converge.

5.4.2 Exploiting Objective Function Structure

The solutions found by CMA-ES are impractical because they take too many evaluations. The *optimize and refine* method and *tube search* force a smoothness constraint on the solution, but these

Profile	Tube Search		Optimize & Refine	
	Fitness	SSE	Fitness	SSE
Mid-latitude Summer	932,322	933	256,605	2,592
Mid-latitude Winter	743,194	738	298,684	3,342
Sub-arctic Summer	760,703	1,610	324,395	3,502
Sub-arctic Winter	1,092,430	348	314,486	1,383
Tropical Summer	1,664,570	1,189	399,106	1,423
Original Profile	1,472,380	1,950	314,486	1,370

Table 5.1: Sum-squared error (SSE) for the *optimize and refine* method and the *tube* search for all the McClatchey profiles we tested.

methods still have difficulty getting close enough to the actual profile to be useful. This section compares several gradient-based methods: the quasi-Newton BFGS, the conjugate-gradient method (CG), and the Levenberg-Marquardt (LM) algorithm. We show that the Levenberg-Marquardt method, which exploits direct knowledge of the objective function's structure, is incredibly effective and efficient on this problem.

We would like to make these results comparable to the results already presented. The problem is, a call to the evaluation function is significantly less complex than a call to the gradient. For example, a call to a finite-difference gradient would require either 43 or 86 calls to the objective function depending on the type of finite-difference method used. Fortunately, the gradient of the temperature problem is not as expensive; logically and empirically, we determined that calling the gradient once is approximately equivalent to calling the evaluation function 20 times. The results present here are, therefore, in computational units. We define a computational unit for the temperature problem as:

$$\text{computational unit} = \text{objective function calls} + 20 \cdot \text{gradient calls}$$

The results of the BFGS, CG, and LM algorithms are very consistent. Like CMA-ES, every trial for both algorithms converges to a solution that is within a tight neighborhood of the optimal given the constraints on the computational units. Figure 5.9 shows the convergence plot for 30 runs of CMA-ES, BFGS, and LM. All the gradient methods find solutions that are satisfactory from a practical point of view.

The most dramatic features in Figure 5.9 is the steep final convergence of the LM algorithm.

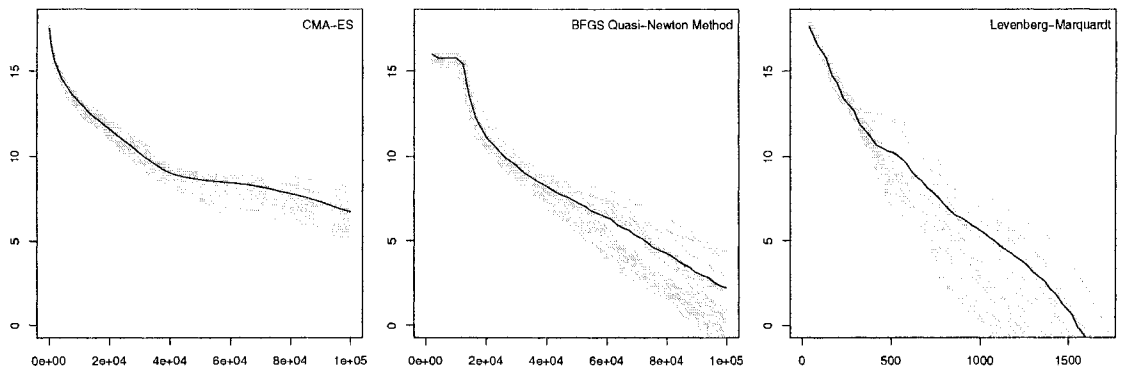


Figure 5.9: Gradient algorithms and the temperature problem. The black line is the average convergence for each algorithm. The convergence for each trial is shown as a gray line and is included for visual variance information. The leftmost graph is an average convergence plot of CMA-ES, included for perspective. The middle graph is a representative convergence graph for BFGS. The rightmost figure is the extremely efficient Levenberg-Marquardt algorithm. Notice that CMA-ES and BFGS have similar scale on the x-axis, which is the number of computational units. The LM algorithm, on the other hand, only uses a little over 1500 computational units to converge.

This is even more extreme because the number of computational units required to converge for LM is only $\approx$ 1500! In contrast, both CMA-ES and BFGS require the full 100,000 computational units, which for CMA-ES are objective function calls only. The efficiency here is a direct consequence of LM exploiting the structure of the objective function. As search approaches the optimal solution, the Hessian approximation gets more and more accurate because the residual term, r_i, gets closer and closer to zero. The Hessian approximation

$$B_k = J(x)^T J(x) \approx J(x)^T J(x) + S(x) = \nabla^2 f(x)$$

gets closer to the true Hessian as the term $S(x)$ goes to zero. This essentially transforms the ill-conditioned temperature ridge into a sphere within a local neighborhood of the optimal solution. And a spherical surface is easy to optimize because every gradient points to the minimum.

The BFGS algorithm is more efficient than CMA-ES, but considering that it is using the gradient, its performance when compared to LM is disappointing. We also ran the conjugate-gradient algorithm and found it was even less efficient. We have already mentioned that the parallel version of CMA-ES would only require 1000 steps to find adequate solutions. This makes the direct search surprisingly competitive on this problem with these well-known gradient algorithms.

5.5 Summary

Temperature retrieval is a real-world optimization problem that has ridges in its search space. Attempts to solve this problem using well-known evolutionary algorithms and local search methods often produce poor results. We know that the algorithms that fail, also do not address the ridge problem. In the previous chapter, we explained why local search and self-adaptation cannot efficiently move along ridges structures. In light of this, it is not surprising that they perform so poorly on the temperature problem. The CMA-ES algorithm has,to a large degree, overcome the problems associated with attempting to adapt rotational parameters. This use of rotated representation has a clear advantage on the temperature problem.

Salomon's *optimize and refine* method and our *tube search* strategy produce approximate solutions quickly, but the temperature profiles found are imprecise. This leaves two limited choices if we are restricting ourselves to direct search methods; either retrieve a quick and dirty solution using a strategy that exploits continuity or spend more time finding an effective temperature profile using an evolutionary algorithm like CMA-ES. Of course, what we would like to have is an algorithm that quickly finds precise solutions. Running CMA-ES with parallel sampling would certainly increase its efficiency.

The Levenberg-Marquardt algorithm utilizes two well-known ideas within the optimization community. First, if gradient information is available, then using it will likely result in a more efficient and effective search. Of course, this not absolute: the performances of the BFGS and conjugate-gradient algorithms were slightly disappointing. Second, the more knowledge you can exploit about a particular problem, the better. This is where the Levenberg-Marquardt gets its efficiency. On temperature retrieval problems, where derivative information is available and the objective function can be stated as a least-squares problem, this algorithm is clearly the best choice.

Chapter 6

Dispersion, Mobility, and Search

It is not clear how global structure impacts search. This is partly because test functions with a less predictable underlying structure are often underrepresented in empirical studies and are overshadowed by easier problems with a unimodal global structure. This creates two problems: First, on a more general level, an algorithm's global search ability can be misunderstood based on how well it solves relatively easy benchmark test problems. For example, solving the Rastrigin function does not necessarily imply that an algorithm will find competitive solutions on an application that is highly multimodal. Second, in order to understand how *global structure* affects search, we must first be able to categorize test functions based on the complexity of their underlying *global structure.*

Ultimately in the chapter, we want to explore the question: how does the more complex global structure, that we know exists in some benchmark test problems, affect evolutionary parameter optimization? We investigate this question from several perspectives.

We start this chapter by introducing an algorithm independent metric that measures *function dispersion* based only on a uniform random sample of points. This allows us to categorize the benchmark test functions described in chapter 3 as either *high dispersion* or *low dispersion* landscapes. At a high level, a low dispersion function corresponds to landscapes that have a *big valley* or *single-funnel.* A high dispersion function, on the other hand, is one where the underlying global structure is not a predictable bowl-like shape. Instead, the best regions of the search space tend to be more spread out.

Then we explore how high dispersion functions impact search. Intuitively, an effective global search algorithm will spend most of its time comparing the best local optima. On high dispersion functions, the best local optima are not be clustered in a compact region of the search space, but exist

in several disperse regions. The way this will impact search is unclear. For example, an algorithms like CMA-ES, where the population stays together and exploits a single region of the search space, may require restarts to visit diverse local optima. CHC's population, on the other hand, can in theory exploit multiple disperse regions of the search space simultaneously. Our hypothesis is that CHC will be a better global search strategy for high dispersion functions. We pose two questions that we believe will help investigate this intuition. How disperse are the local optima visited during a search? How does the dispersion of local optima relate to performance?

We explore these questions using a new metric that quantifies *algorithms dispersion*, which we define as the dispersion of local optima visited during search. We call this metric *mobility*. Then we compare local search, CHC and CMA-ES, on a set of high dispersion test functions and show that, in low dimensions, algorithms that visit more disperse local optima (higher mobility) tend to find better overall solutions on high dispersion functions. In general, we find that the heuristics employed by CHC create higher mobility, and therefore, render CHC more effective. However, as the dimensionality increases, this trend no longer holds. This is partly because the differences in performance between these algorithms is less significant (see chapter 3) and partly because our mobility metric does not work as intended in high dimensional space.

The CMA-ES algorithm is really employing "local search" heuristics to the global search domain, and therefore, must rely on restarts to increase its mobility. This emphasis on detecting and exploiting local structure also makes CMA-ES competitive on low dispersion multimodal functions. Hansen and Kern have empirically shown that CMA-ES performs well on multimodal functions where there exists a "global topology" and found that CMA-ES is more prone to fail when the global structure is less predictable [41].

We believe that dispersion will capture this notion of "global topology". Consequently, we investigate how dispersion impacts CMA-ES. First, we show that our dispersion metric can predict how well CMA-ES will perform on multimodal problems. Our results indicate that CMA-ES works very well on low dispersion functions, but has difficulties when compared with high dispersion functions. Then we explore the question: why does CMA-ES behave inefficiently on problems that do not have a globally convex structure? Our research indicates that the adaptive step-size heuristic, called *cumulation*, does not function as intended when the best regions of the search space are too

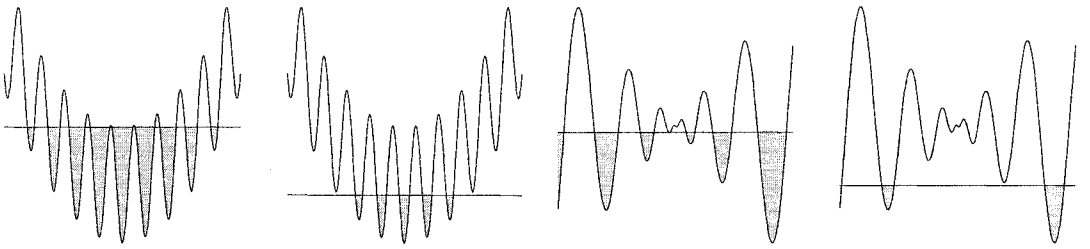


Figure 6.1: The graphically illustration of dispersion as a means of measuring global structure. Decrease the threshold will tend to decrease the dispersion of the best regions of the search space on the Rastrigin function (left) and increase the dispersion of the Schwefel function (right).

spread out.

6.1 Function Dispersion

The goal in this section is to introduce an algorithm independent metric, which we call *dispersion*, that distinguishes functions based on their underlying global structure. Our metric exploits the following observation: the sub-threshold regions of problems that have a convex global structure will tend to get closer to one another as we decrease the threshold, while the opposite is true on functions where the global structure is less predicable. We expect the sub-threshold regions on these functions to be more disperse as we decrease the threshold.

The one dimensional Rastrigin and Schwefel functions shown in Figure 6.1 graphically illustrate this idea. The shaded area in each figure represents the sub-threshold regions of the search space. As we decrease the threshold, the distance between the distinct basins of attractions that form below the threshold changes. Notice that on the Rastrigin function, the best regions of the search space tend to get closer as we decrease the threshold. On the Schwefel function, however, decreasing the threshold actually increases the distance between the basins of attraction. In other words, lowering the threshold will cause the dispersion of the best regions of the search space to increase.

We create sub-threshold regions of the search space by selecting the best points of a random sample. The sample of points is constructed such that it represents an aspiration threshold expressed as a target percentage of the search space. This allows us to use the dispersion metric in two ways.

1. We can compare the dispersion of different functions at a particular threshold.

2. We can calculate the dispersion of a single function at different thresholds.

In the second case, we might measure the dispersion of the points estimated to be in the top 5% of the search space and compare this to points estimated to be in the top 0.5% of the search space. A *low dispersion* function is one where the dispersion decreases as the sample is restricted to better regions of the search space. This corresponds to functions with a convex global structure (e.g. Rastrigin's function). A *high dispersion* function is one where dispersion stays constant or increases as the sample is restricted to better regions of the search space. Functions of this type have a less predictable global structure (e.g. Schwefel's function).

The dispersion metric gives us a means to determine a significant feature of the global topology of a function. Are the best regions in the search space localized or disperse? Some highly multimodal functions have dispersion measures that are approximately the same as a simple unimodal sphere function: when this is the case, empirically experiments confirm that low dispersion functions are easy to optimize.

There exists other recently introduced metrics that attempt to measure similar trends in the global topology, but these measures have the unrealistic requirement that the search space be enumerated, or at least that the best local optima are located and enumerated. These metrics are briefly reviewed later in this section.

We calculate dispersion by drawing a uniform random sample of points from the search space and calculate approximate thresholds that breaks the sample into subsets representing the best X% of the search space. For example, if we sample 100,000 random points, we can define a 10% threshold to be the maximum fitness of the best 10,000 points. A 1% threshold is the maximum fitness of the best 1,000 points. As the percentage decreases, the threshold also decreases. In this way, a threshold is determined by a sample size, s_v, and a percentage p. At this threshold, *dispersion* refers to the average pair-wise distance between the best $p \times s_v$ points in the sample. In practice, computing the average pair-wise distance of $p \times s_v$ points can be computationally costly. For example, if $s_v = 100,000$ and $p = 0.01$, then $s_v \times p = 1000$. This implies computing the distance between $1,000^2/2 - 1,000 = 499,000$ points.

To make the dispersion calculation more efficient, we define $s_b = s_v \times p$ to be a fixed sample size. Our dispersion metric is: given an objective function, $f(\vec{x})$, a fixed sample size, s_b, and a variable sample size, s_v, *dispersion*(s_b,s_v,$f(\vec{x})$) calculates the average pair-wise distance between

the best s_b points of the random sample, s_v. The variable sample size, s_v can be expressed as a function of the fix sample size s_b and the desired percentage, p: $s_v = s_b/p$. For example, given a fixed sample size of $s_b = 100$, a variable sample of size $s_v = 100/0.01 = 10,000$ is needed to create a $p = 0.01$ threshold. The dispersion pseudo-code is given in Figure 6.2 and emphasizes clarity instead of efficiency.

Dispersion(s_b,s_v,$f(\vec{x})$)

---

input
Integer s_b – the fixed sample size
Integer s_v – the variable sample size
$f(\vec{x})$ – the objective fitness function

variables
allPoints – a vector of random $\{\vec{x}, f(\vec{x})\}$ pairs.
bestPoints – a vector of the best $\{\vec{x}, f(\vec{x})\}$ pairs.

for $i = 1$ to s_v **do**
  Create a random point, $\vec{x}$.
  Evaluate its *fitness*, $f(\vec{x})$.
  Add $\{\vec{x}, f(\vec{x})\}$ to allPoints.
end for

bestPoints $\leftarrow$ best s_b points of allPoints
return *average pairwise distance*(bestPoints)

Figure 6.2: The *dispersion* pseudo-code.

6.1.1 Benchmark Test Functions

The point estimate of dispersion given in the dispersion pseudo-code is not invariant with respect to scale. That is, it is possible for two functions with completely different underlying structures to have similar *dispersion* measures for a given sample size. However, decreasing the aspiration threshold will change this point estimate, and this change is a more accurate measure of the underlying topology. We calculated the dispersion of several classic benchmark test functions discussed in chapter 3. In particular, we computed the dispersion of the best $s_b = 100$ points for a set of increasing sample sizes, starting with $s_v = 100 \cdot 2^0$, and doubling the sample size each iteration until we reached $s_v = 100 \cdot 2^{12}$ samples (e.g. 100, 200, 400, 800, ..., 204800, 409600). This yields

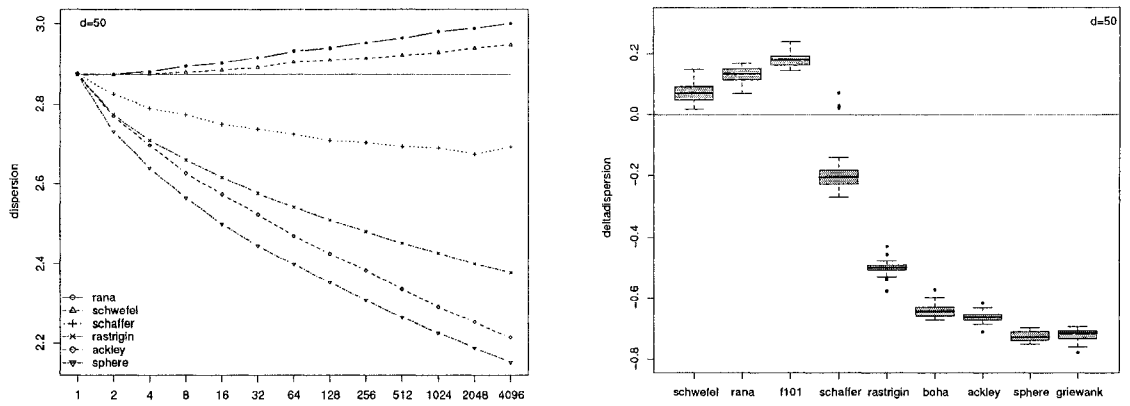


Figure 6.3: The dispersion of each test function computed for different sample sizes ($\times 100$). On the left most graph, the horizontal indicates the dispersion of the initial random sample of 100 points (denoted *dispersion*$(100, 100)$). Because we rescaled the boundaries of each function such that the range was $(0, 1)$, the initial dispersion using the first 100 points is the same for all functions. The right most graph shows the change in dispersion of each test function.

both a point estimate of dispersion at a particular threshold as well as information about how the dispersion changes as the threshold decreases. In order to make cross-function comparisons more meaningful, we rescaled the best 100 points based on the boundaries of the function such that the new minimum and maximum parameter values are 0 and 1 respectively. Then, we computed the distance for all $(100^2/2 - 100) = 4,900$ pairs of points and averaged this number. We repeated this 30 times for each function in 20, 50, and 100 dimensions. Figure 6.3 shows the average dispersion as a function of sample size for several 50 dimensional functions. The solid horizontal line indicates the dispersion of a random sample of 100 points, which we denote *dispersion*$(100, 100)$ (leaving the objective function out of the notation in Figure 6.2). We found that as we increased the sample size, the dispersion increased on the Schwefel, Rana and F101 functions. The opposite was true for all the remaining functions. Increasing the sample size actually decreased the dispersion of these functions.

We computed the change in dispersion for each benchmark function by subtracting the lowest threshold dispersion value in our set from the average dispersion value when no selection is applied. Specifically, we computed the value: *dispersion*$(100, 409600)$ - *dispersion*$(100, 100)$. On functions where the dispersion increases as we decrease the threshold (by increasing the sample size s_v), we should get a positive number. When the dispersion decreases as we decrease the threshold, we will get a negative number. Figure 6.3 also shows the change in dispersion for several benchmark test

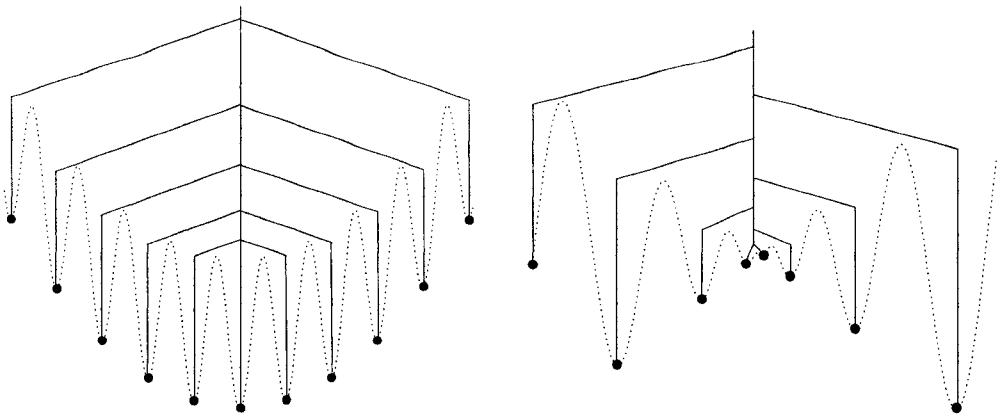


Figure 6.4: The disconnectivity graph for the Rastrigin (left) and Schwefel (right) functions.

problems. This allows us to clearly distinguish between *low* and *high* dispersion functions. For the remainder of the paper, low dispersion is associated with functions whose change in dispersion is negative. Low dispersion functions include the Schaffer, Rastrigin, Bohachevsky, Ackley, and Griewank functions. The Schwefel, Rana, and F101 are referred to as high dispersion functions.

6.1.2 Related Work

Doye and Wales use disconnectivity graphs to visualize the global structure of various energy landscapes [28]. The disconnectivity graph is actually a tree whose leaves are the local optima. Two or more leaf nodes are connected by a node if they exist in the same basin of attraction. Doye uses different energy levels to define basins of attraction. A split in the tree implies that every local optima below this point is reachable by a path whose fitness does not exceed the given energy threshold. In other words, the minimum *saddle point* of the path connecting two local optima is lower than the energy threshold. Figure 6.4 shows the disconnectivity graphs for the one dimensional Rastrigin and Schwefel functions. Notice that the Rastrigin function has a single dominate stem. This is indicative of a "big valley" topology. On the other hand, the Schwefel function has several stems that split early at high energy levels and has a less predictable global structure.

Disconnectivity graphs have been applied to discrete optimization problems where landscape concepts, such as local optima, basins of attraction, and saddle points are clearly defined [33]. Flamm *et al.* extend these critical definitions so that disconnectivity graphs, which they call *barrier trees*, can be generated for highly degenerate discrete problems [33].

Locatelli *et al.* offers a different view of global structure [55]. A graph is defined where the

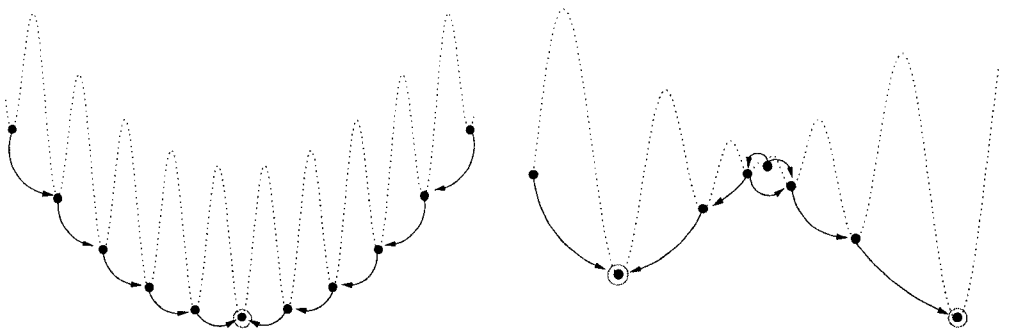


Figure 6.5: Locatelli's view of a funnel landscape.

nodes of the graph are the local optima of the function and an directed edge is extended from node X to node Y if $f(X) > f(Y)$ and X and Y are adjacent local optima. Local optima that are within distance r of each other are said to be adjacent. A funnel bottom is a node with no outgoing directed edges. An example of this graph for the one dimension Rastrigin and Schwefel functions is given in figure 6.5. One drawback to this method is that Locatelli's graphs will change dramatically depending on what distance measure r is used.

One problem with *disconnectivity graphs*, *barrier trees*, and Locatelli's method is that they all requires locating the relevant local minima in the search space. This cannot be efficiently done for "black box" optimization problems where the underlying mathematical function is unknown or does not exist. Even when derivative information exists and can be used to catalog all the *local optima* and *saddle points*, these methods are limited to relatively small problems. Doye *et al.* admit that finding all the minima for large clusters is impossible: "We therefore stopped searching once we were confident that we had obtained an accurate representation of the low-energy regions of the [Potential Energy Surface] " [26]. Hallam and Prügel-Bennett point out that exhaustive enumeration of the search space raises concerns regarding the usefulness of *barrier trees* [38]. To mitigate this problem, they use a branch and bound technique to find only the best local optima. But this technique is still limited to relatively small problems. Hallam and Prügel-Bennett construct barrier trees for MAX-SAT problems with 40 variables.

Our *dispersion* metric is appealing because it requires a relatively modest number of samples to observe a noticeable change in function dispersion, even on problems with 500 parameters. Furthermore, *disconnectivity graphs* and *barrier trees* do not directly address the *distance* between local optima but rather the *barrier height* between local optima.

6.2 Mobility: A Measure of Algorithm Dispersion

Our dispersion metric clearly brands the benchmark test functions based on their underlying structure. The distinction creates an opportunity to better understand the global search properties of heuristic search. In particular, we explore in this section how high dispersion impacts CHC, CMA-ES, and local search.

It is reasonable to expect that an effective global search will spend most of its time comparing the best local optima. On a high dispersion function, this implies that search should be comparing different and disperse regions of the search space, and then focusing its resources on those areas that appear to be the most promising. In order to quantify this intuition, we define a new metric, *mobility*, that quantifies *algorithm dispersion*, which we define to be the dispersion of local optima visited during search. This allows us to explore two questions: First, how disperse are the local optima visited during a search? Second, do effective global search algorithms tend to be more mobile? We show that on two difficult multi-modal, high-dispersion functions, algorithms that visit more disperse local optima tend to find better overall solutions in low dimensions.

6.2.1 Measuring Mobility

Our goal is to measure the dispersion of local optima visited during search. In order to measure how many of the local optima were sampled during search,we need quantify what it means to be "close" to a local optima. Once we know which local optima were sampled, we need to define a metric to measure their spread.

Two problems arise with respect to measuring the number local optima sampled during search. First, how do we define boundaries for the basins that surround local optima. Second, once we have defined this boundary, how do we cluster points that fall within a given basin?

Ideally, we would look at the basins of attraction that contain the best points of the search space. We have already discussed one way of doing this; choose a fitness *threshold*, and only consider points that fall below this value. This defines a measure of closeness to the local optima that is based on fitness and only includes the best local optima of the search space.

Like our dispersion metric, we need to define an appropriate threshold value for each function. In low dimensions, we can define this value by enumerating a fine mesh grid in the search space,

sorting the points by fitness, and retrieving the maximum fitness of the best percentage of points. For example, we may want to find the threshold that defines the best 10% of the search space. This estimation is only an approximation due to finite precision effects. In higher dimensions (more than three), the enumeration and evaluation is intractable so another method must be used.

We defined a threshold as follows: First we ran local search, CHC, and CMA-ES on the Rana and Schwefel problems described in chapter 3. Each algorithm ran for 30 trials and each trial ran for exactly 25,000 and 50,000 evaluations in five and ten dimensions respectively. Since parameter settings affect performance, we ran two versions of each algorithm. Local search and CHC were run using 10-bits and 20-bits of precision. As in previous chapters, CHC uses the time-tested population size of 50. We ran CMA-ES with rank-μ-updates and a population size of 200 and 500. We distinguish each algorithm based on its parameters; CHC-10, CHC-20, LS-10, LS-20, CMA-200, and CMA-500. We used *restarts* to ensure that each algorithm used all the allotted evaluations. That is, if an algorithm finished before all the evaluations were used, the algorithm was given a random restart during the same trial and allowed to run until it uses up its remaining evaluations.

Then we combined all the points visited, from all 30 trial of each algorithm, and choose the threshold to be the fitness of the best 20% of the points. In other words, we looked at every point evaluated during 30 trials of CHC, CMA-ES, and local search, sorted the points and picked the threshold value to be the maximum fitness of the best 20% of all the points. We use this aggregate number to evaluate the global search properties of each trial.

Once a threshold has been calculated, we consider only the unique points that are below threshold. We assume that these points fall into the basins of attraction for the local optima we are interested in counting.

Once we have determined these points, we constructed a completely connected graph of the local optima with edges weighted by Euclidean distance. We describe how we assign each sub-threshold point to a unique local optima later in this section. From this, we created a minimum spanning tree of the local optima. Figure 6.6 shows an example of the minimum spanning tree for local optima found by CHC on the Rana function in two dimensions. An algorithm's *mobility* is the sum of all the edge weights from the minimal spanning tree.

Mobility measures the minimum connecting distances between a set of points, and does not

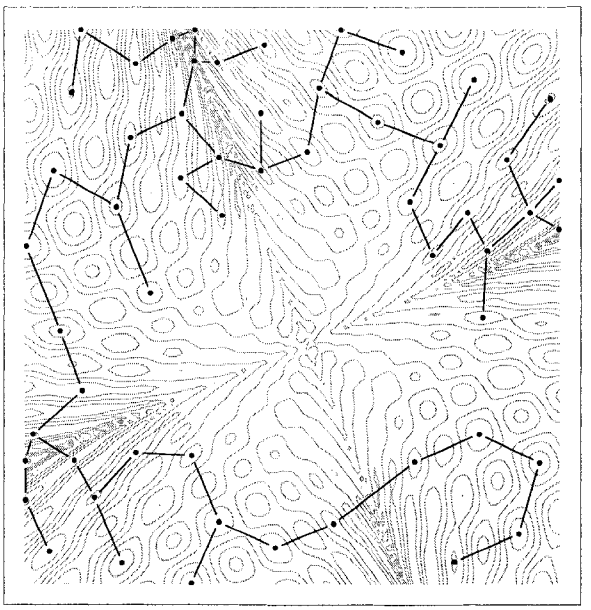


Figure 6.6: A minimal spanning tree of local optima on the Rana function.

distinguish, for example, between four points connected on a line and four points on connected on a box with equal edge lengths (Figure 6.7). In this example the *mobility* metric has a value of 3 in both configurations. We believe mobility gives a rough estimate of the dispersiveness of the local optima visited during search.

6.2.2 Related Work

Schuurmans and Southey defined three metrics that characterized local search performance on satisfiability problems [87]. The first metric, *depth*, measures the average depth local search travels into the fitness function. More specifically, *depth*, refers to the number of unsatisfied clauses at any point during search. *Mobility* measures how rapidly local search moves away from recently explored areas. Finally, *coverage* captures how well local search is exploring new regions that have not been previously visited. Schuurmans and Southey show that effective local search algorithms tend to have low depth and high mobility and coverage.

We are defining and exploring similar properties in the parameter optimization domain. Schuurmans' depth measurement is similar to our threshold. Our mobility metric is similar to both Schuurmans' mobility and coverage. Schuurman and Southey used *Hamming* distance to compute their metrics. In the parameter optimization domain, *Euclidean* distance is more meaningful. Like

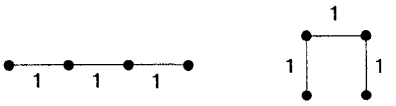


Figure 6.7: The mobility metric is invariant under certain sets of patterns.

Schuurmans and Southey, we relate mobility to algorithm effectiveness.

Bit-encoded algorithms can explore a large portion of the search space by only changing a few bits. One concern is: does measuring Euclidean mobility bias our results to bit-encoded algorithms? Although a small change in Hamming distance *can* correspond to a large change in Euclidean space, mobility is only measuring the distance to those sub-threshold neighbors. In other words, local search visiting a distant neighbor, as guided by its heuristic, does not imply that the point will be included in the mobility measurement. As discussed earlier, this point is considerably stronger when the test functions are non-symmetric and non-separable. We also observe that Hamming-space mobility would mean something completely different, which renders a direct comparison in Euclidean space confusing and potentially impossible.

6.2.3 Computing Mobility

In low dimensions, we use a modified form of local search to assign each sub-threshold point to a unique local optima. The neighborhood pattern of our modified local search was created by modifying the current best solution by ± 0.001 in each dimension. This creates a compact neighborhood that includes two points in each dimension that surround the current best point. Keeping the neighbors close to the sub-threshold point is important for two reasons. First, it ensures that each point is assigned to the correct local optima. That is, local search can't discover better basins farther away using a small step-size. Second, as already discussed in chapter 4, a high precision local search also decreases the number of false local optima generated by ridges in the search space.

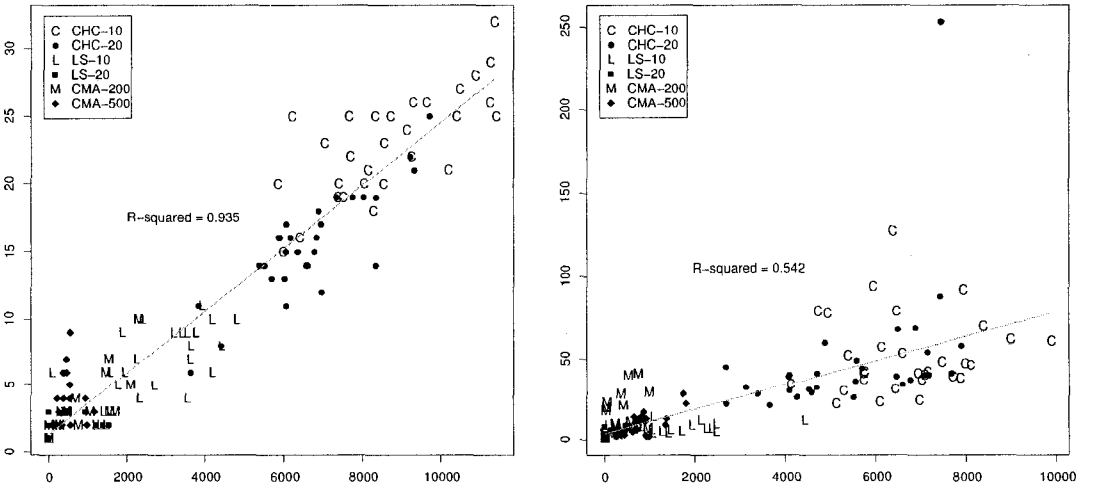
Various measurements for both the 5-D Rana and Schwefel functions are listed in Table 6.1. For each function, CHC-10 and CHC-20 visit significantly more basins of attraction and have significantly higher mobility than either local search (LS-10 and LS-20) or CMA-ES (CMA-200 and CMA-500). LS-10 had significantly higher mobility and visited more basins than the 20-bit local search. On the Rana Function, CMA-500's higher mobility was significant when compared with CMA-200. There were no other statistical significant patterns found between the algorithms.

	Rana		Schwefel	
	Mobility	Basins	Mobility	Basins
CHC-10	6,667.6	52.2	8,702.9	23.1
CHC-20	5,423.1	48.4	6,632.8	15.5
LS-10	995.3	6.47	2,530.9	6.17
LS-20	4.72	1.87	169.0	1.4
CMA-200	277.6	10.9	492.9	2.43
CMA-500	712.3	8.43	348.8	3.1

Table 6.1: Average *mobility* and number of *basins* visited per trial after 25,000 evaluations on the Rana and Schwefel 5D functions.

	Rana Fitness	Schwefel Fitness
Best known	-2,651	-2,371
CHC-10	-2,411.9	-2,334.4
CHC-20	-2,370.9	-2,274.9
LS-10	-2,010.6	-2,015.0
LS-20	-1,614.1	-1,790.4
CMA-200	-1,991.0	-1,906.2
CMA-500	-2,085.5	-1,884.0

Table 6.2: Average *fitness* per trial after 25,000 evaluations on the Rana and Schwefel 5D functions.



(a) Schwefel (b) Rana

Figure 6.8: The number of basins visited (local optima) vs. Mobility on the Schwefel and Rana 5D test. Algorithms that visit more local optima tend to have a higher mobility and vice versa.

In five dimensions we noticed a high correlation between the number of basins visited and the mobility. Intuitively, this makes sense. Algorithms that visit more local optima will tend to have higher mobility and vice versa. Figure 6.8 shows the number of basins vs. the mobility on the Schwefel five dimension test. The Schwefel test has a strong correlation of 0.94. The correlation on the Rana five dimension test was 0.55.

In higher dimensions, performing modified local search from each of the threshold points becomes extremely computationally intensive. The primary problem here is that the modified local search simply takes too long to converge from the sub-threshold points to the local optima we wish to count.

This requires us to calculate the mobility metric discussed in the previous section without explicitly knowing to which basin of attraction each sub-threshold point belongs. We did this by constructing a minimal spanning tree of all the sub-threshold points. In other words, all the sub-threshold points will now be included as nodes of the minimal spanning tree. The mobility metric is different from the previous mobility metric in several ways. Most importantly, the nodes of the tree are no longer local optima, but rather sub-threshold points. As a result, the edges that join them will be included in the mobility measurement. This mobility metric is still the sum of the edge lengths in the tree, but will now include the edges between points that lie in the same basin. This means the mobility measure will be slightly inflated because points within the same local optima will now add to the mobility metric. Comparing Figure 6.6 with Figure 6.9 graphically explains this difference.

In ten dimensions, we calculated each algorithms mobility after 10,000 and 50,000 evaluations. The ten dimensional results for both the Rana and Schwefel functions are listed in Table 6.3. After 10,000 evaluations, CHC-10 and CHC-20 had significantly higher mobility than the other algorithms on both the Rana and Schwefel functions. Similarly, local search (LS-10 and LS-20) had higher mobility than CMA-ES (CMA-200 and CMA-500). After 50,000 evaluations, CHC-10 still exhibits significantly higher mobility than all the other algorithms on the Schwefel function. However, the difference between CHC (CHC-10 and CHC-20) and CMA-ES (CMA-200 and CMA-500) on Rana after 50,000 evaluations was not significant. All four algorithms outperformed local search (LS-10 and LS-20) on both functions after 50,000 evaluations.

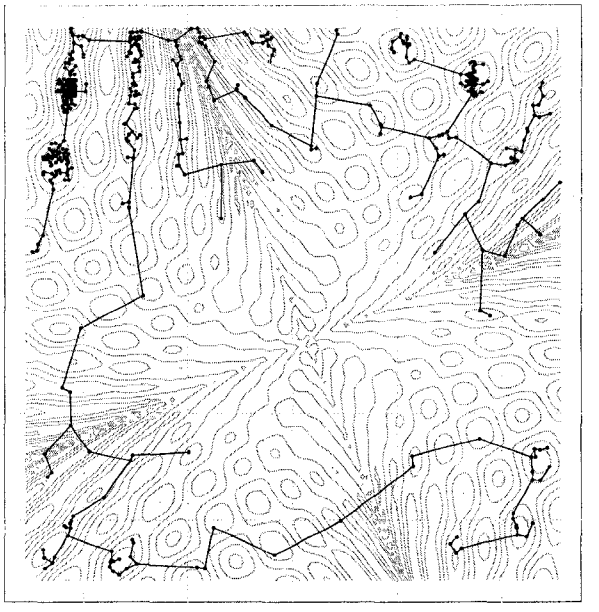


Figure 6.9: A minimal spanning tree of all the sub-threshold points on the Rana function. The minimal spanning tree in Figure 6.6 was constructed using the same sub-threshold points as a staring position for local search.

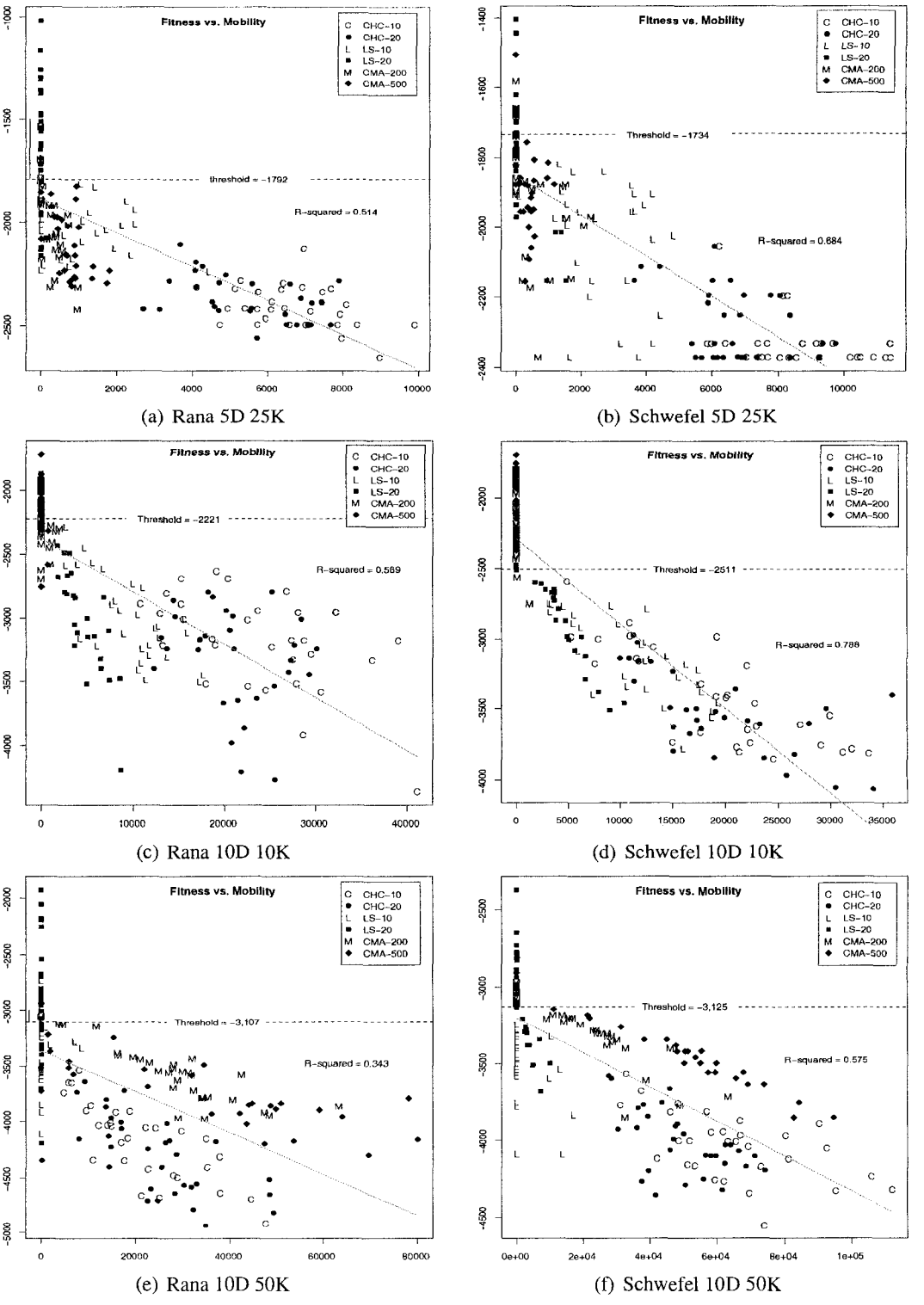
6.2.4 Mobility and Performance

Is mobility a good measure of global search performance? To answer this question we looked at the correlation between average fitness and mobility for each trial. Figure 6.10 shows the general trend for all six tests. Notice that algorithms with higher mobility tend to have more effective solutions.

There is one exception to this observation. Some trials have no mobility, but vary in solution quality. In other words, each graph in figure 6.10 has trials that have a large range of fitness values,

	Rana		Schwefel	
	10K	50K	10K	50K
CHC-10	23,793.1	21,316.8	19,179.7	65,009.2
CHC-20	21,068.3	24,951.0	19,433.4	49,907.7
LS-10	9,819.6	835.5	11,512.6	2,137.4
LS-20	3,709.0	0.0	3,330.5	1,609.4
CMA-200	255.7	28,249.0	45.8	20,973.0
CMA-500	48.1	32,475.9	0.0	43,249.8

Table 6.3: Average *mobility* after 10,000 and 50,000 evaluations on the Rana and Schwefel 10D functions.



(a) Rana 5D 25K

(b) Schwefel 5D 25K

(c) Rana 10D 10K

(d) Schwefel 10D 10K

(e) Rana 10D 50K

(f) Schwefel 10D 50K

Figure 6.10: Mean Fitness vs. Mobility. In each case, there is some correlation that indicates more fit trials had high mobility. Intuitively, we would expect algorithms that explore more to find better solutions.

	Rana		Schwefel	
	10K	50K	10K	50K
Best Known	-4,933.0	-4,933.0	-4,556.9	-4,556.9
CHC-10	-3,184.2	-4,177.8	-3,411.6	-4,038.7
CHC-20	-3,319.0	-4,313.3	-3,523.5	-4,013.1
LS-10	-3,011.8	-3,382.5	-3,143.2	-3,464.0
LS-20	-2,822.2	-2,958.3	-2,749.3	-3,083.7
CMA-200	-2,253.4	-3,574.0	-2,159.3	-3,285.3
CMA-500	-2,124.5	-3,757.2	-1,989.3	-3,386.3

Table 6.4: Average *fitness* after 10,000 and 50,000 evaluations on the Rana and Schwefel 10D functions.

yet have zero mobility. These are the "stacked" points in the leftmost position of the graphs.

There are two reasons why trials could have low mobility and a large range of fitness values. Recall that both of the mobility metrics defined in this paper only include points below a given threshold value, which is computed based on competing fitness values among all trials, across all algorithms. In order for a trial to have mobility, it must have some points below the threshold. So, trials that have no mobility and poor fitness simply did not visit below threshold basins of attraction. Notice that many of the low mobility points lie above the fitness threshold.

On the other hand, some algorithms tend to have low mobility and rather effective solutions. This is most pronounced in Local search (LS-10 and LS-20) after 50,000 evaluation in 10 dimensions (figures 6.10(e) and 6.10(f)). One explanation for this is that local search visited a few good basins of attraction that were probably relatively close together and failed to explore other below threshold areas of the search space. This could happen, for example, on a long ridge where local search creeps toward good solutions. In this case, local search uses many evaluations exploring only a small part of the space.

6.2.5 Discussion

In ten dimensions, local search is significantly more mobile during the 10,000 evaluation measurement, but falls off the map entirely after 50,000 evaluations. At the same time, CMA-ES has more mobility after 50,000 evaluations. Longer running snapshots have higher thresholds. As algorithms

begin to converge, densely populated sub-threshold distributions lower the threshold. So, early on in the search, the distribution of CMA-ES is still quite large and unfocused. Local search, on the other hand, has already focused in on several local optima. Eventually, CMA-ES starts to form a compact (and densely populated) distribution that focuses in on one or more local optima. This effectively lowers the threshold to a point that nearly excludes local search. Notice that CHC remains fairly consistent in during the 10,000 and 50,000 evaluation mobility snapshots.

In all but the Rana ten dimensional, 50,000 evaluations, test, CHC had significantly higher mobility. Why does CHC, on average, tend to visit more of the search space? Perhaps some of the answer lies in CHC's distribution. CHC's crossover operator HUX, or Half Uniform Crossover, exchanges half the non-matching bits. One consequence of this, according to Eschelman, is that children are always maximum Hamming distance from their two parents [32]. This highly disruptive operator tends to spread search out. It is possible that the distributed nature of CHC's child distribution, at least to some degree, accounts for its high mobility.

Restarts also help algorithms with stale distributions explore new parts of the search space. *Cataclysmic mutation*, for example, forces CHC to look in other parts of the search space once the population begins to converge. In the five dimensional tests, the lower precision search was significantly more mobile than its high precision counterpart in every case. This held in ten dimensions for the 10,000 evaluations tests, where local search actually had some mobility. The high precision local search has twice as many neighbors in each step, which quickly burns up valuable evaluation calls. And all of the neighbors are within the low precision neighborhood, which, in terms of exploration, adds nothing to the search. Furthermore, the small step size allows the high precision search to creep on ridges where the low precision search gets stuck and is forced to restart.

As we increase dimensionality, the trend between mobility and algorithm performance weakens. One reason for this may be that Euclidean distances in high dimensional space render our mobility metric less precise than in five and ten dimensions. We have also noticed that as we increase dimensionality, the difference in algorithm performance becomes less noticeable on high-dispersion functions. In chapter 3 we noticed that there was no significant difference in algorithm performance on the rotated versions of the 20-dimensional Rana and Schwefel functions. If there is a trend that exists in higher dimensional space between mobility and performance, we have not been able to

detect it experimentally.

6.3 Dispersion and CMA-ES

We know that CMA-ES tends to exploit one region at a time because it is primarily employing local search heuristics. This means it must rely in restarts to increase its mobility. Hansen and Kern [41] studied CMA-ES on multimodal functions and state that, "If the local optima can be interpreted as a perturbations of an underlying unimodal function, then larger populations can detect this global topology". Furthermore, they continue, "A strong asymmetry of the underlying function jeopardizes a successful detection and can lead to failure". In other words, the performance of CMA-ES is greatly impacted by the underlying structure. We proposal that dispersion is one way to quantify and identify functions with underlying unimodal surfaces that CMA-ES will do well on. We also predict that "asymmetric" functions will tend to have higher dispersion.

The distribution used by CMA-ES is initially isotropic. As a result, the initial value of σ is critical for exploration. Hansen and Ostermeier suggest that the quality of solutions found by CMA-ES using a small initial step-size are often determined by the location of the starting point [43]. Hansen and Kern further emphasize that small initial step-sizes can have a considerable impact on the performance of CMA-ES[41]; they set the σ_0 between 20% and 40% of the length of the constrained region of the search space. Auger and Hansen also test CMA-ES on multimodal functions and set $\sigma_0 = 50\%$ of the constrained region [7]. Hansen *et al.* summarizes that "A larger step-size improves the global search performance of a local search procedure"[43]. In this section, we use an initial step-size suggested by Auger and Hansen which is $\sigma_0 = (U - L)/2$, where L and U are the lower and upper bounds of the constrained search space.

As discussed in chapter 3, each test function is usually bound constrained. As implemented here, the test functions have a boundary penalty to insure that a solution is in the feasible region. This method, described by Hansen *et al.* [41], is the standard way of handling boundary conditions when testing CMA-ES on multimodal functions [41; 7]. The penalty is proportional to the number of parameters that fall outside the boundary. Therefore, the penalty for a point where one parameter is outside the feasible region is less than a point where all the parameters are outside the feasible region.

Hansen and Kern [41] summarize that a larger population size considerably improves the performance of CMA-ES on all problems with one notable exception, the Griewank function. Here, they found that a larger population performed better in lower dimensions, but a smaller population was more effective in higher dimensions. In chapter 3, we already discussed one reason this is true: the Griewank function actually gets easier as dimensionality increases [109]. We can also use dispersion to explain this apparent anomaly. In low dimensions (e.g. two), the dispersion of the Griewank function is much higher than the sphere function. However, referring back to Figure 6.3, we can see that the dispersion of Griewank and the sphere are nearly indistinguishable. This is not a shortcoming of our dispersion metric, but rather an indication of how smooth the high dimensional Griewank actually is.

In order to understand the relationship between function dispersion and the performance of CMA-ES, we ran CMA-ES with several different population sizes on some of the benchmark test functions described earlier. Our hypothesis is that CMA-ES will perform relatively well on the test functions that we categorized as *low dispersion* and will be less effective on the *high dispersion* functions.

Figure 6.11 shows the convergence of CMA-ES on six of the test functions. In every case, we used restarts to ensure that each instance of CMA-ES ran for exactly $200,000$ evaluations. The population size was varied using $\lambda = 50, 100, 250$, and 500. On all the *high dispersion* functions, the larger values of λ produced the most effective solutions. However, none of these represent solutions that are close to the optimal. This is consistent with the observations made by Hansen *et al.*[41]; when the underlying problem structure is less predicable, the performance of CMA-ES suffers. CMA-ES was much more successful on the *low dispersion* functions. The Schaffer and Bohachevsky were easily solved for all values of λ. On Rastrigin's function, CMA-ES was more successful with larger values of λ, but was able to get relatively close to the optimal solution even for small population sizes. Referring back at Figure 6.3, we can see that dispersion gives a reasonable prediction of how well CMA-ES will perform on multimodal surfaces.

In the conclusion of their paper, Hansen and Kern suggest starting with a small population and increasing its value with each restart of CMA-ES [41]. Auger and Hansen have implemented this strategy, which they refer to as IPOPCMA-ES, where the population doubles each time a restart

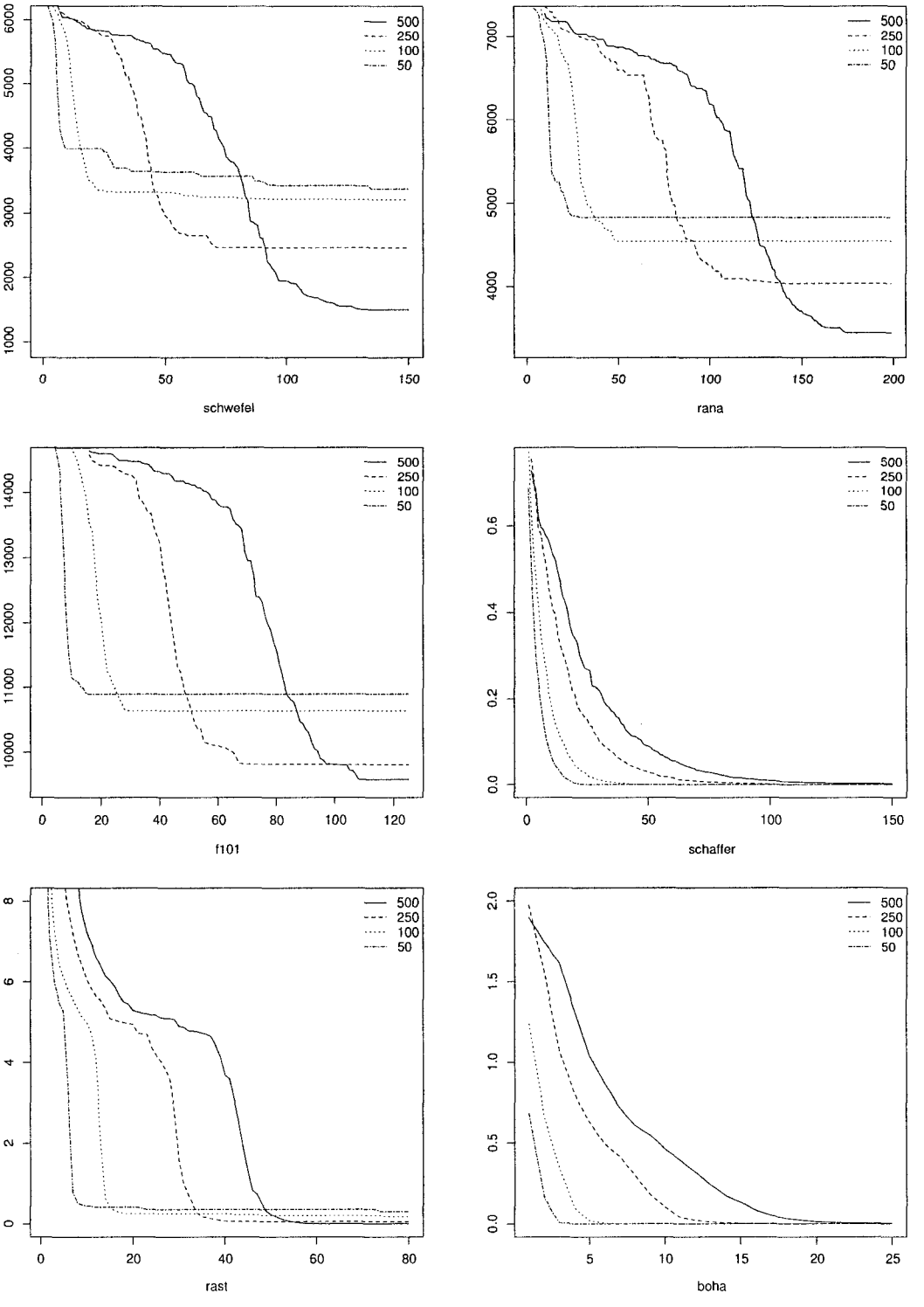


Figure 6.11: The median fitness vs. number of evaluations ($\times 1000$) for CMA-ES on several 20D benchmark test functions. The larger population size is the most effective on the *high dispersion* functions, which includes Schwefel, F101, and Rana. This is also true for the *low dispersion* and highly multimodal Rastrigin function. The *low dispersion* Shaffer and Bohachevsky functions are easily solved for all values of λ.

occurs [7]. Starting with the default population size of $\lambda = 4+3 \cdot log(n)$, each restart was initialized with a population size twice that of the previous instance. For example, in 20 dimensions, the population sequence would be $12, 24, 48, 96, 193, 348, ...$ The authors assert that this essentially makes CMA-ES parameter-free because the population size does not need to be tuned for each problem [7].

Looking at the convergence graphs for the Shaffer and Bohachevsky functions in Figure 6.11, it is not too surprising that this type of strategy would perform well on these functions where smaller population sizes are equally effective, and therefore, preferred because they have a faster rate of convergence. But when the test suite contains mostly *unimodal* and *low dispersion* multimodal functions, this average behavior can be misleading. On *high dispersion* functions, incrementally increasing the population size incurs dramatic increase in convergence time. For example, Hansen *et al.* found that CMA-ES used over 250,000 evaluations in order to solve the 10 dimensional Schwefel function. Similarly, Auger and Hansen found that when they increased the maximum number of evaluations from $300,000$ to $900,000$ the performance of the 30 dimensional Rastrigin function greatly increased.

One conclusion here is that on *high dispersion* functions, where larger values of λ are likely to be the most effective, IPOPCMA will require an unrealistic number of evaluations just to start using a more effective value for λ. While this efficiency may be satisfactory on some problems, it infeasible for others.

6.3.1 Understanding Rate of Convergence

On high dispersion functions, the performance of CMA-ES can sometimes be quite poor; the solutions are less effective when compared to other functions and there is a noticeable increase in the overall efficiency. Intuitively, it makes sense that finding the global optimum of a high dimension, high dispersion function is difficult. There are simply too may "funnels" that can trap even the most effective search algorithm. On low dispersion functions, like Rastrigin's, every starting position is in the only (and optimal) funnel.

Although we can reason why CMA-ES is less effective, there does not appear to be an easy explanation as to why it is also less efficient in terms of convergence. Figure 6.12 shows the nor-

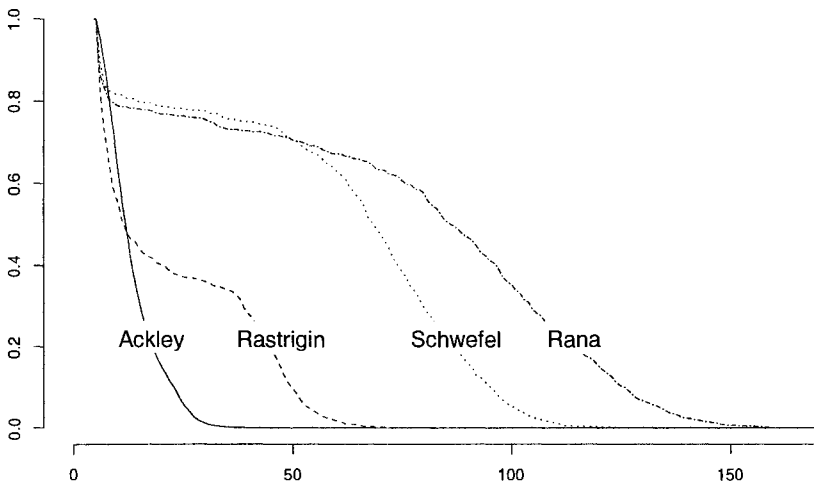


Figure 6.12: Scaled fitness vs. number of evaluations ($\times 1000$) for select functions.

malized mean convergence of CMA-ES with a population size of $\lambda = 500$ on four 20 dimensional benchmark test functions: Ackley, Rastrigin, Schwefel, and the Rana function. Notice that for the high dispersion Rana and Schwefel functions, it takes well over 100,000 evaluations to converge. Although $\lambda = 500$ seems too large for a 20 dimension problem, we have already seen that the best solutions on high dispersion functions are likely to be discovered with larger populations.

One reason the convergence is slow is that CMA-ES tends to "waste" evaluations on points that are infeasible, even on low dispersion functions like Rastrigin and Ackley. A points is infeasible if any parameter falls outside the bounds of the search space. Since CMA-ES is more effective using larger initial σ values, most of the points in the initial population will fall outside the constrained boundary on high dimensional problems. These evaluations are not wasted in the sense that they are not useful, but they cannot offer any improvement and can, therefore, be seen as inefficient. On average, over 10% of the $200,000$ evaluations used on the 20D Rana and Schwefel functions were infeasible. This is almost three times higher than that of the low dispersion functions we tested. This means that CMA-ES requires almost $20,000$ evaluations just to find the boundaries of the problem.

With a large initial step size (σ), all of the initial sample points are outside the boundary of the feasible region. This is confusing because the sample distribution should at least create some points near the center of the mean, which is chosen in the feasible region. To make sense of this, consider the following. We created 500 Gaussian distributed points around the origin with a step size of $\sigma = 1$. This simulates the initial population that CMA-ES generates around a random

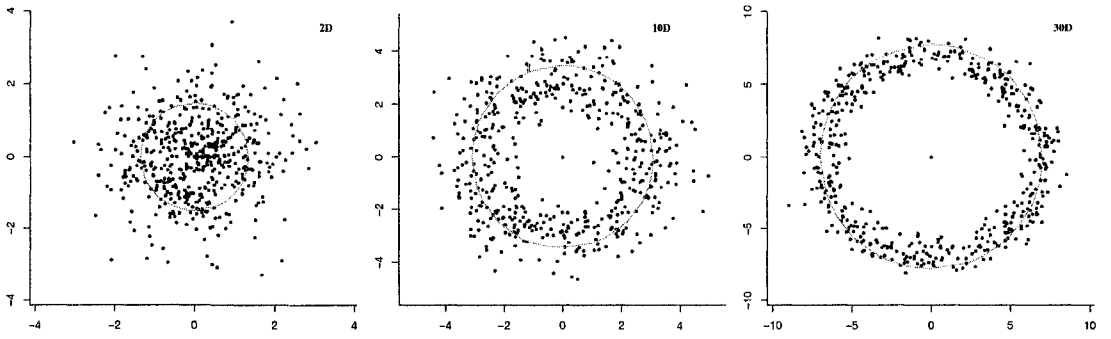


Figure 6.13: As the dimensionality increases, the variance of the Gaussian distribution becomes more constant.

starting point. If we look at any 2D slice of the data, we see that some points are close to the mean while other are more distant. Unfortunately, (and surprisingly) as the dimensionality increase, the distance between the mean and the closest point becomes very similar to that of the mean and the most distant point. To visualize this, we computed the Euclidean distance from the mean to every point sampled. We plotted the polar coordinates of this distance and a random angle. Figure 6.13 shows that as the dimensionality increases, the variance of the distribution becomes more constant and larger. This means that higher dimension will tend to sample hyperspheres rather than Gaussian distributions [9].

One enticing potential remedy to this problem is to simply re-sample each infeasible point until it becomes feasible. But Figure 6.13 also shows that in high dimensions, sampling a point in the feasible space using a large σ is too improbable to be a practical solution. Unfortunately, decreasing σ will also decrease the effectiveness of CMA-ES.

A high percentage of infeasible offspring does not sufficiently account for the inefficient behavior of CMA-ES on high dispersion functions because this also occurs on *low* dispersion functions. Recall that CMA-ES uses a distinct global step size, σ, to appropriately scale its covariance matrix distribution. Hansen and Ostermier state that this is necessary for two reasons [43]. First, the learning rate on the covariance matrix is too slow. Second, the optimal step length cannot be approximated well by the covariance matrix alone. This is why the actual sample population created by the distribution defined by the covariance matrix is rescaled based on the current step size, σ. When search is making progress, the evolution path is longer than expected, and σ grows. On the sphere function, for example, σ will increase as CMA-ES approaches the minima during exploration. Once

the algorithm begins to exploit a local optima, the evolution path is likely to be smaller, and σ will shrink. This creates a region of higher density during exploitation.

We looked at the values of σ as a function of time on the standard test functions described earlier. The step-size adaptation on the low dispersion functions behaved much like that of the sphere. This is not too surprising because we know that the underlying global structure is unimodal. However, we found that on the higher dispersion functions, the step-size grew rather large, on average between two and three times its initial value. At the same time, the actual distribution of offspring was decreasing. This has two important implications. First, the distribution of the covariance matrix is responsible for the decrease in actual sample distribution. This is a concern because Hansen *et al.* asserts that the distribution defined by the covariance matrix is sub-optimal in value and often too slow [43]. Second, if σ is steadily increasing, the cumulative path length is remaining above its expected length. This implies that the adaptive *cumulation* heuristic cannot decide when to stop exploring.

6.4 Summary

In this chapter, we have defined a new metric, *function dispersion* that distinguishes a function based on its *global structure*. On low dispersion functions, the best regions of the search space become more localized as the sample size increases (threshold decreases). The opposite is true for high dispersions functions; as the sample size increases, the best regions of the search space tend to become more disperse.

Mobility measures algorithm dispersion, which is the quality and dispersion of the local optima visited during search. As with our function dispersion metric, defining thresholds creates basins of attraction that are close to local optima in terms of fitness. Additionally, it focuses our study on only the best local optima. High precision local search provides a reliable way of clustering points in the same basin of attraction in low dimensions. The minimal spanning tree metric, *mobility*, is highly correlated with the number of optima found in a search. In high dimensions, estimating mobility gives a reliable measure of algorithm dispersion—the spread of local optima visited during search. In either case, algorithms with higher mobility tend to have more effective solutions in low dimensions.

In higher dimensions, this trend is difficult to capture. Some researchers have noticed that CMA-ES is inefficient and ineffective on multimodal functions where the underlying structure of the problem is unpredictable. Dispersion is one way of quantifying global structure. This allows us use dispersion to predict how efficient and effective CMA-ES will be.

We have identified two reasons why CMA-ES is less efficient on *high dispersion* functions. First, many of the initial offspring are not in the feasible region. CMA-ES can waste up to 20% of its evaluations simply finding the boundaries of the problem. This is really a constraint handling problem, yet it is not clear how to address this issue in order to improve the efficiency of CMA-ES. Second, and probably more serious, we found that the step size adaptation mechanism does not work as expected on high dispersion functions. Instead, the distribution defined by the covariance matrix is primarily responsible for the decrease in the actual distributions that creates the offspring.

Detecting function dispersion is a logical step towards creating algorithms that work well on a variety of functions. Looking at how the multiple instances interact is one promising way of identifying the dispersion of the function during search. If the instances are getting close, then increasing the population of a single instance and terminating others may be an effective strategy.

Chapter 7

Double Trouble: How Funnel Landscape Characteristics Impact Search

We have already seen that many artificial test functions have a "big valley" topology, where a decrease in fitness implies that, on average, search is getting closer to the global optimum. Although the search space is highly multi-modal, the local optima are structured such that there exists a global trend toward the best solution. The underlying *global structure* of this type of problem is roughly unimodal.

While low dispersion landscapes may be common, there also exist several real-world applications that do not have a unimodal underlying global structure. Wales [99] suggests that many optimization problems in computational biology are difficult because local optima often form in distinct, spatially separate clusters within the search space. Problems of this type have multiple funnels, resulting in a landscape that has a less predictable underlying global structure. Furthermore, the most difficult problems arise when the optimal configuration is "hidden" in a funnel that is proportionally smaller than the other funnels in the search space.

Researchers have found that the most effective search strategy for these difficult instances is to quickly sample the bottom of as many funnels as possible. Algorithms such as *basin-hopping* [100; 101] and *local optima smoothing* [2] are able to explore a particular funnel by descending down a sequence of local optima to the bottom of a funnel. By comparing the best solutions in each funnel, they increase their chance of finding the best overall configuration.

We have found that evolutionary algorithms do not perform well on applications where the optimal funnel is proportionally smaller than other funnels in the search space. This is partly because

many evolutionary algorithms initially *explore* the search space and then *exploit* the promising regions (or region) identified during the exploration process. Intuitively, this makes sense: if an algorithm does not sample the entire search space first, its overall effectiveness may be largely determined by where it is initialized. The assumption here is that exploration, using a limited number of samples, can distinguish between different regions of the search space based on average effectiveness. This simple model has proven to be an effective strategy on the standard set of artificial test functions described in chapter 3.

The way that global structure impacts search is not well understood, partly because, as the previous chapter shows, many of the test functions used for evaluation have single-funnel, low dispersion landscapes. There are also a few high dispersion test functions that have multiple funnels, but the number of funnels increases with dimensionality. This creates a gap in our ability to evaluate global structure. That is, either we have a single funnel or we have a complex surface that often contains several funnels. This makes it difficult to understand search behavior in high dimensions.

We have several objectives in this chapter. First, we describe the Lennard-Jones cluster problem and discuss the algorithms that can locate the global optimum on these problems. Then we empirically show that CMA-ES, CHC, and Local Search do not perform well on this application when the optimal funnel occupies a small proportion of the search space.

Second, we describe a method for creating landscapes that contain exactly two funnels, regardless of the problem size. The characteristics of these double-funnel problems, such as funnel size and depth are adjustable. Then we empirically examine how the characteristics of this landscape impact CHC and CMA-ES. We find that both algorithms have an extremely low probability of success when the global optimum is located in a proportionally smaller funnel. Our results suggest that the exploration process is more biased toward the relative size of the funnels and not their overall quality.

Finally, we observe from this that too much exploration *can* hinder search in high dimensions. We reduce the amount of exploration in CSA-ES by using a variable population size. This strategy results in a performance gain on the double-funnel functions that are the most difficult for CMA-ES and CHC. One counter-intuitive result of this research is that limiting the degree to which an algorithm explores the search space can actually improve its global search performance.

7.1 Background and Motivation

There are several characteristics that make real-world optimization problems difficult. The degree to which an algorithm will perform well on an application partly depends on how well the algorithm can deal with the features that make the problem difficult. For example, the Rastrigin function represents a classic single-funnel (or "big valley") landscape. It is comprised of a cosine term and a quadratic sphere; the sphere gives the function its global structure and the cosine term creates local optima.

$$\begin{aligned} f_{\text{Rastrigin}} &= \sum_{i=1}^{N} x_i^2 + 10 \sum_{i=1}^{N} (1 - cos(2\pi x_i)) \\ &= \text{global structure} + \text{local optima} \end{aligned}$$

There are at least two ways to deal with the large number of local optima ($\approx 10^N$) that make this problem difficult [50]. The first is to exploit the problem's *separability*, which we have discussed can reduce problem difficulty to N one-dimensional lines searches. The other strategy is to exploit the problem's global structure. Like Basin-Hopping [100] and Local Optima Smoothing [2], CMA-ES avoids getting trapped in local optima by exploiting an underlying structure. The main question we are exploring in this chapter is: how does this underlying structure impact evolutionary search?

In this section, we describe the Lennard-Jones optimization problem and discuss why researchers believe some instances of this problem are difficult. Then we describe how these problems are best solved and compare this with the results of several evolutionary algorithms.

7.1.1 Lennard-Jones Clusters

Researchers within the computational chemistry community have started to pay attention to how global structure affects problem difficulty [51]. Much of their attention has been devoted to studying Lennard-Jones clusters, which are a class of configuration optimization problems where the goal is to find the spatial positions for a set of atoms that has the smallest potential energy. The potential energy of the system is based on the distance between all the molecules of the cluster. The Lennard-Jones potential is

$$E = 4\epsilon \sum_{i<j}^{N} \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right)$$

where r_{ij} is the Euclidean distance between the centers of atoms i and j, and N is the number of atoms in the cluster. For simplicity, $\epsilon = \sigma = 1$. This potential results in molecular clusters with compact geometries [27].

The energy surface of the Lennard-Jones potential is highly multimodal and the number of local optima increases with problem size. This means that multiple restarts of *local search* will become a less effective global optimization strategy as the number of the atoms in the cluster increases. Fortunately, the local optima are not randomly distributed in the search space; some Lennard-Jones instances also have a convex *global structure*, or *single funnel*, where there exists a monotonically descending sequences of successively adjacent local minima [25].

Any algorithm that can avoid getting trapped by local optima should be able to find the global optima. From this perspective, finding the best solution for larger problems should be more difficult because the dimensionality is greater and there are more local optima. However, there are several Lennard-Jones instances where finding the optimal configuration of a small cluster requires significantly more effort than finding the global solution for clusters with a greater number of atoms. For example, the optimal configuration of the $N = 38$ atom Lennard-Jones problem is more difficult to find than the global optima of other instances as high as $N = 60$ atoms, despite the fact that the $N = 38$ has considerably fewer local optima and object parameters. One explanation for this discrepancy in difficulty is that the $N = 38$ atom instance has a less predictable underlying *global structure* and that the best configuration exists in a funnel that occupies a smaller proportion of the search space. Assuming that a search algorithm has the ability to move between local optima, the underlying *global structure* of a problem may have a greater impact on problem difficulty than the number of local optima [77].

Figure 7.1 shows a cartoon of the 38-atom problem. The optimal solution has a much different configuration than the next best local optima, meaning that they are distant from each other in Euclidean space. There is also a high fitness barrier that divides the two solutions, creating a double-funnel landscape. The funnel containing the optimal solution is proportionally smaller than the sub-optimal funnel.

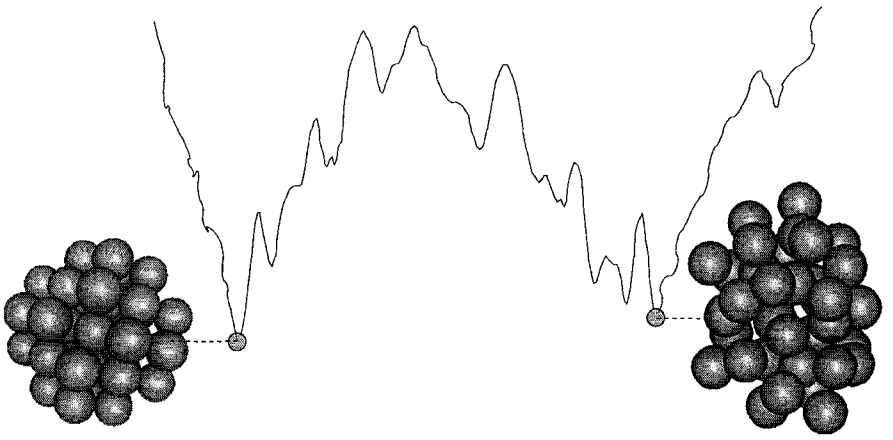


Figure 7.1: A cartoon of the 38-atom Lennard-Jones Cluster instance. The optimal configuration (left) is distant from the second best solution in the search space (right). Moving from the right configuration to the optimal would require traveling over a large fitness barrier.

7.1.2 Basin Hopping

The *basin hopping* [100; 101] algorithm was motivated by two simple observations. First, and most general, escaping local optima is difficult because the fitness of an uphill step is greater than the fitness of the local optima. One solution to this well-known observation is to sometimes accept a non-improving move, an idea first proposed by Metropolis [65]. Second, gradient-based search is ineffective on funnel landscapes because they have a large number of local optima. However, since the local optima are clustered together such that there exists a monotonic sequence of decreasing adjacent local optima that eventually reach the bottom of the funnel, hopping between these adjacent local optima is one way of exploiting this global structure.

Basin hopping is a point-based method, where new candidate solutions are created by randomly perturbing the current solution based on a *step-size*. After each step, the fitness of the two points are compared. If the new point offers improvement or equal fitness, the step is accepted. If the new point is less effective, then the non-improving move is sometimes accepted based the relative fitness of each point. This type of search method is often referred to as a Monte Carlo strategy.

The value assigned to each candidate solution is not based on their location in space, but rather the value of the local optima that is obtained by applying local minimization. Figure 7.2 displays the plateau surface that results from this implicit transformation. Each point is assigned the value of the local optimum that exists in the same basin of attraction. One consequence of this is that there

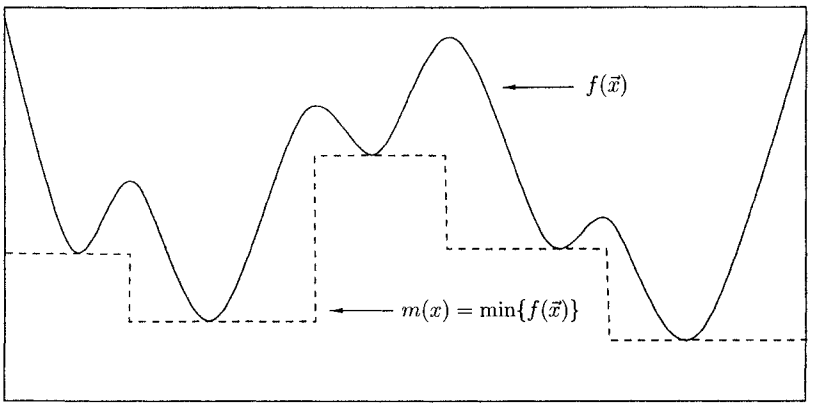


Figure 7.2: The transformation of the fitness function used by Basin Hopping. The fitness of each point in the search space is not based on its location, $f(x)$, but rather the value found by local minimization from that point, $m(x)$.

are no fitness barriers between local optima. An uphill step on $f(x)$ constitutes a plateau step on the transformed surface, $m(x)$. A downhill step on $m(x)$ implies that search has found an improving local optima. This also means that the Metropolis criteria is used to accept non-improving moves is actually accepting non-improving local optima. This combination of Monte Carlo search plus fitness transformation has been successful in locating the optimal solutions to many problems that have a predictable underlying global structure (e.g. single-funnel).

The step-size adaptation is simple: if the current trial point is accepted, the step-size is increased and decreased otherwise. If a uniform distribution is used to displace each coordinate when perturbing the best solution, then the resulting step-sizes are on the order of those found by Wales and Doye [100]. However, we found that if a Gaussian distribution is used, then the adapted step-size values are about half that of the values obtained using a uniform distribution.

Leary [51; 52] introduced a greedy basin hopping algorithm with a constant step-size, which he called "Monotonic Basin Hopping", or MBH. He found the resulting algorithm could locate the difficult global optimum on problems where the original version failed. Essentially, MBH is a basin hopping algorithm that only accepts improving moves. Leary also fixed the step-size value to $\sigma = 0.21$. Leary comments that "this parameter can be adaptively adjusted as the algorithm proceeds", but no adaption mechanisms was proposed. In our experience, the high selective pressure of the greedy algorithm causes the step-size to quickly converge to zero when using the original basin hopping adaptation scheme.

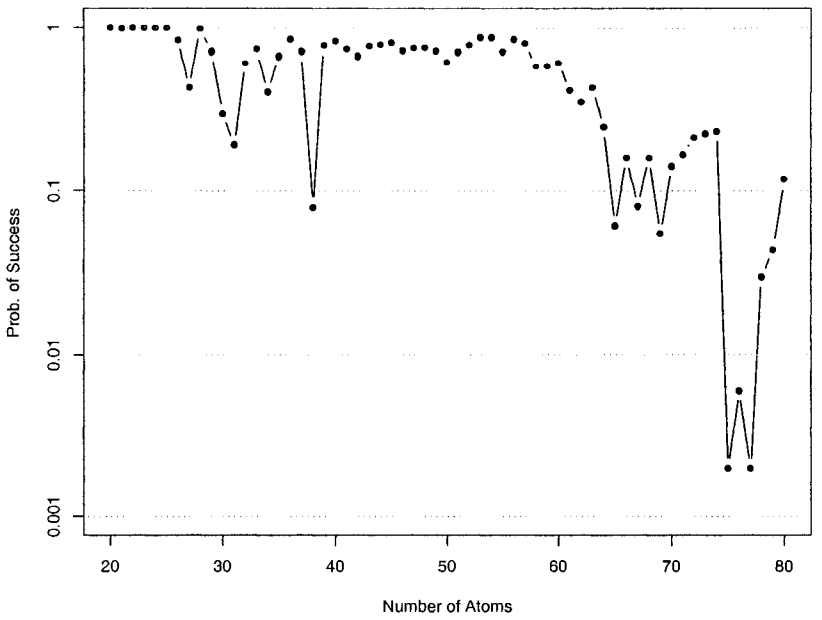


Figure 7.3: The probability of success based on 1000 trials of MBH. Each trial took exactly 1000 steps. This is a replication of Figure 3 produced by Leary using our implementation of MBH.

Figure 7.3 shows the probability that MBH succeeds on each instance of Lennard-Jones from $N = 20 - 80$ atoms. The probabilities are based on 1000 trials. This is a replication of Figure 3 produced by Leary [51]. MBH finds the optimal solution to the $N = 38$ atom test problem $80/1000$ trials. This is slightly lower than that of the original BH algorithm, which successfully locates the optimal solution around 160 times in 1000 trials. For the $N = 75$ test problem, MBH find the optimal solution only twice in 1000 trials. BH never finds the optimal for this instance.

There are two reasons why MBH could fail to find the global optimum: It could have entered the wrong funnel, or it could have used all of its allocated trials before finding the correct configuration. The optimization community is starting to realize that funnel size is an important characteristic in problem difficulty. On the $N = 38$ atom instance, it is estimated that the optimal funnel occupies about 10% of the entire search space.

7.1.3 Evolutionary Algorithm Performance

We ran CMA-ES, CHC, and local search on select Lennard-Jones cluster problems. CMA-ES and local search used *random restarts*. CHC used its soft restart mechanism. Each algorithm was run for 30 trials, where each trial was allocated $500,000$ evaluations. For CMA-ES, we used two population sizes: $\lambda = 200$ and $\lambda = 500$. CHC and local search each used 10- and 20-bits of precision. As in

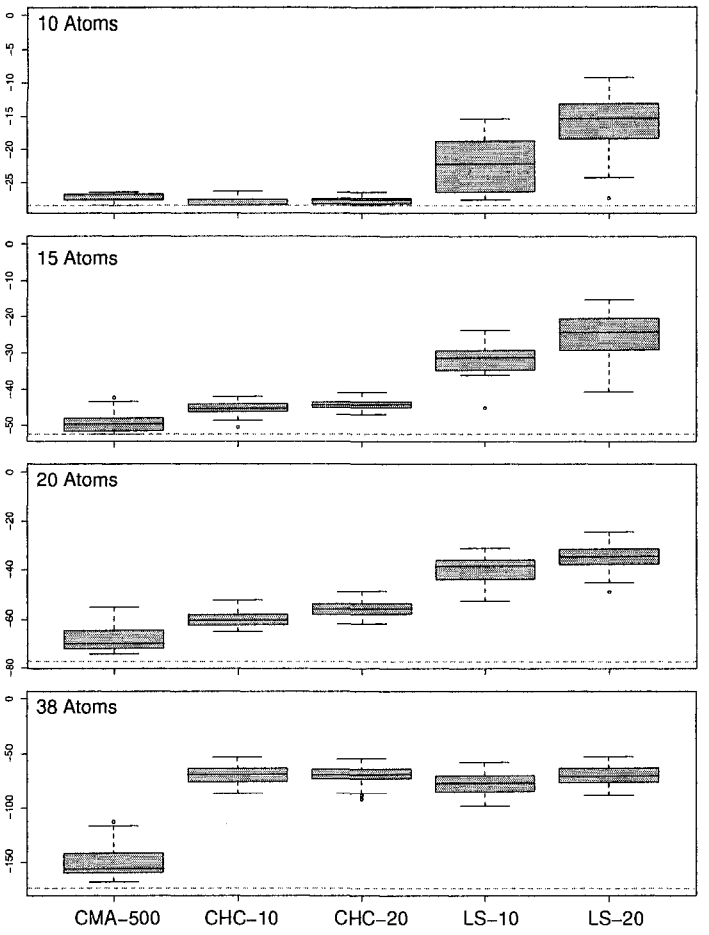


Figure 7.4: Evolutionary algorithms on select Lennard-Jones cluster problems. The dashed line is the optimal objective function value. Notice that none of the algorithms solved the 20- or 38-atom problems.

previous chapters, each algorithm is differentiated based on its parameters: CMA-200, CMA-500, CHC-10, CHC-20, LS-10, and LS-20. Four problems were considered: 10, 15, 20, and 38 atoms. Box-plots of the objective function values for the best solution found during each trial, and for each algorithm, are displayed in Figure 7.4. CMA-ES using a population size of 500 was always more effective than CMA-ES using $\lambda = 200$. For clarity, we omit the CMA-200 results.

CHC is the most consistent and effective on the 10-atom problem. On all other problem instances (15, 20, and 38 atoms), CMA-ES finds the best solutions when compared with local search and CHC. Unfortunately, none of the 30 trials used by CMA-ES find the optimal solution for either the 20- or 38-atom problem. The concern here is that the dimensionality is too large and more evaluations are needed to sample the optimal funnel.

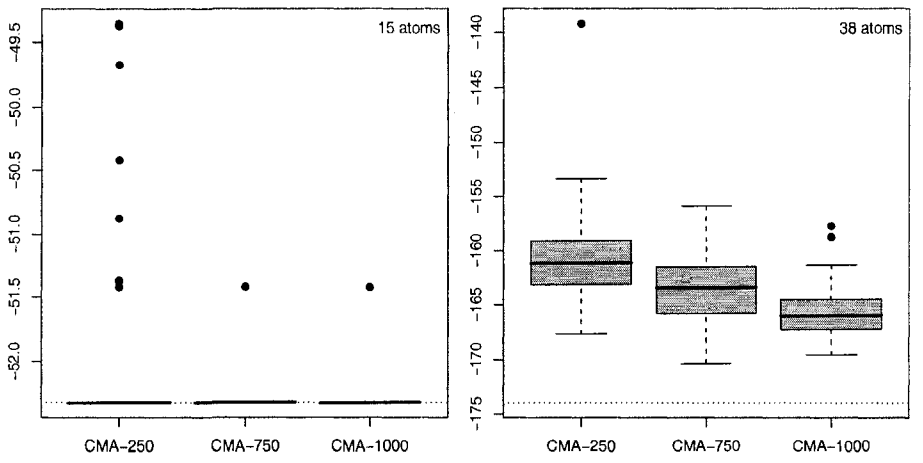


Figure 7.5: Increasing the population size and the number of evaluations results in a more effective search with CMA-ES. However, even with the increased number of evaluations, CMA-ES still does not find the optimal solution for the 38−atom Lennard-Jones problem.

We ran CMA-ES again on the 15- and 38-atom problem, this time allocating 100 trials and allowing each trial to run for 2.5e6 evaluations (5× longer). We also ran CMA-ES with $\lambda = 250$, 500, and 1000. Figure 7.5 shows our results. On the 15-atom instance, CMA-ES consistently finds the optimal solution. This is a positive result because this problem has a single funnel. Notice that the smaller population size of $\lambda = 250$ gets trapped more often in sub-optimal configurations. This is consistent with our expectations of CMA-ES on multimodal, single-funnel functions. Larger populations are also more effective on the 38-atom instance. However, none of the trials found the best configuration for this problem. We conjecture that the large initial step-size recommended for multimodal surfaces tends to pull CMA-ES into the larger, yet sub-optimal, funnel. The remainder of this chapter explores this hypothesis.

7.2 Creating Double-Funnel Landscapes

We have already seen that high dispersion test functions have a multi-funnel global structure. For example, the Schwefel, Rana, and Whitley [109] functions all have several funnels. Aside from their complexity in higher dimensions, these test functions also have the optimal solutions on, or near, the boundary of the search space. This means that an algorithm's performance may largely depend on how the boundary conditions are handled. We propose two test functions where the best solutions are not on the boundary of the search space and that only contain two funnels, regardless

of the problem size. The relative size and depth of each funnel is also adjustable, making it possible to understand how funnel characteristics impact search.

In this section, we describe two double-funnel test problems. First, we create a simple surface comprised of two quadratic spheres. Then, we take this simple surface and add local optima to it. This creates a multi-funnel surface similar to Rastrigin's function.

7.2.1 The Double-Sphere

The landscape structure of our simple *double-sphere* test function is the minimum of two quadratic functions, where each sphere creates a single funnel in the search space. The placement of each sphere is critical because the barrier that divides them will be inconsequential if they are too close. We also want the relative barrier between each funnel to scale with dimensionality. This means that the distances between the center of each funnel should also increase as dimensionality increases.

To address these concerns, we place each quadratic sphere along the positive diagonal of the search space, which is bounded on the interval $[-5, 5]$. The optimal sphere is located in the middle of the positive quadrant of the search space, at $\mu_1 = 2.5$ in each dimension. The sub-optimal sphere is centered at $\mu_2 = -2.5$ across all dimensions. The distance between each funnel increases proportionally with dimensionality, and this construction creates an underlying surface that is *globally* non-separable.

Lennard-Jones double-funnel problems are difficult when 1) the sub-optimal funnels is nearly as deep as the optima funnel, and 2) the basin of attraction to the optimal funnel is small. For example, the optimal funnel for the 38-atom Lennard-Jones problem is believed to be about 10% of the size of the sub-optimal funnel and the best solution in the sub-optimal funnel differs from the global optima by only about small degree [27]. We estimate that the depths of the funnels differ by about 8% with respect to the minimum barrier that divides them.

We simulate this by increasing the height of the sub-optimal funnel by a value of d. That way, the value of the optimal funnel is unchanged. In order to change the relative size of each funnel, we scaled the sub-optimal funnel by a constant factor, denoted s. This way, the optimal funnel retains it shape regardless of scaling, and therefore, has a more consistent level of difficulty. Multiplying the sub-optimal funnel by a number greater than one will create a more narrow sub-optimal funnel.

The opposite is true when s is less than one. The overall form of our multi-funnel sphere function is:

$$f_{\text{double-sphere}}(\vec{x}) = \min\left(\sum_{i=1}^{N}(x_i-\mu_1)^2, \quad d\cdot N + s\cdot\sum_{i=1}^{N}(x_i-\mu_2)^2\right)$$

In order to make s the primary control characteristic for the size of each basin of attraction, we shifted the mean of the sub-optimal sphere such that the barrier between them, which is the point at which they intersect, is always located at the origin of the search space. This configuration requires the mean of the sub-optimal sphere to be $\mu_2 = -\sqrt{(\mu_1^2 - d)/s}$. This places some limitations on our choices of d and s. As d approaches μ_1^2, the problem becomes unimodal. If s is too small, we push μ_2 outside the bounds of the search space.

By shifting μ_2, we also know the minimum barrier height between the two spheres will occur at the origin of the search space and that it will have a value of $f_{\text{double-sphere}}(\vec{0}) = N\cdot\mu_1^2$. The sub-optimal sphere always has an optimal objective function value of $f_{\text{double-sphere}}(\vec{\mu_2}) = d\cdot N$ and the optimal sphere has an objective function value of zero. The objective function difference between the minimum barrier height and the optimal funnel is $x_o = N\cdot\mu_1^2$. The objective function difference between the sub-optimal funnel and the minimum barrier height is $x_1 = N(\cdot\mu_1^2 - d)$. When we compute the percent difference between x_0 and x_1, which is

$$\%\text{ difference} = \frac{d}{(\mu_1^2 - 0.5d)},$$

we confirm that the barrier height that divides the two funnels is independent of N, and that it only depends on the height parameter of the sub-optimal sphere, d. For values of $d = 1, 2, 3$, and 4, the sub-optimal funnel is $\approx$ 17%, 38%, 63%, and 94% different, in terms of objective function value, from the minimum height that divides the two funnels. A value of $d = 1$ does not seems unrealistic when comparing this number to the percent difference estimated for the 38-atom Lennard-Jones instance (8%).

The values for s and d control the relative size and depth of the sub-optimal funnel. The leftmost graph in Figure 7.6 is a diagonal slice showing how the different values of d impact the depth of the sub-optimal funnel. The middle and right-most contour plots illustrate the impact of s. The two funnels are the same size in the middle graph (e.g. $s = 1.0$), but the right-most graph creates a larger sub-optimal funnel (white) using $s = 0.7$.

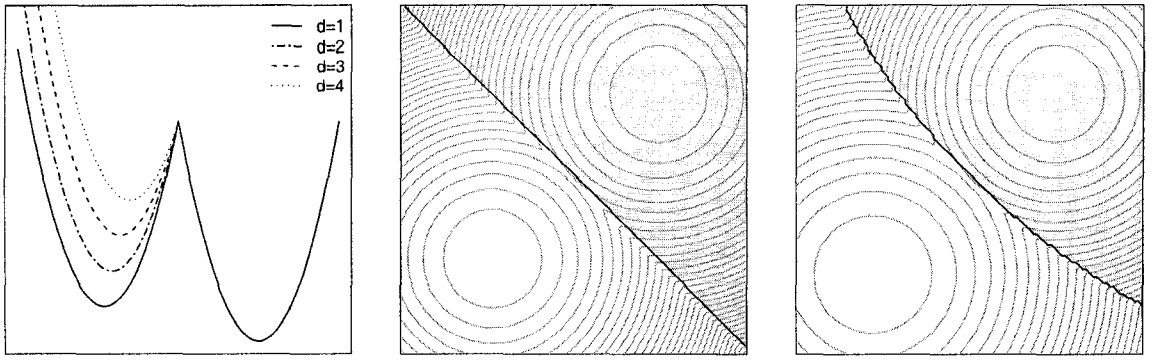


Figure 7.6: The impact of d and s on the *double-sphere* function. Increasing d creates more distinction between the funnels (left). When $s = 0$ (middle), the two funnels are the same size. Decreasing s creates a larger sub-optimal funnel (right).

7.2.2 The Double-Rastrigin

We wanted a double-funnel test problem with properties similar to Rastrigin's function because it would isolate global structure as the main difference impacting problem difficulty on a problem that is well-understood. We create a double-funnel version of Rastrigin's function by adding local optima to the *double-sphere* function. We translate the cosine term used in Rastrigin's function by μ_1 so that the minimum of the local optima component is centered at the bottom of the optimal funnel. The local optima function is:

$$h(\vec{x}) = 10 \sum_{i=1}^{N} (1 - \cos 2\pi(x_i - \mu 1))$$

The overall form of the *double-Rastrigin* function is the sum of the *double-sphere*, which gives the problem its global structure, and the local optima, $h(\vec{x})$.

$$f_{\text{double-Rastrigin}}(\vec{x}) = f_{\text{double-Sphere}}(\vec{x}) + h(\vec{x})$$

One-dimensional diagonal slices of both Rastrigin's original function and our new double-funnel Rastrigin function are displayed in Figure 7.7. The local optima term is bounded between $[0, 20N]$, since the cosine function can generate values between $[-1, 1]$. On average, it adds $10N$ to each to the value of the double-sphere. If we assume that the average barrier height is now $N(\mu_1^2 + 10)$, then the percent difference between the minimum barrier height and each local optima changes to:

$$\%\text{ difference} = \frac{d}{((\mu_1^2 + 10) - 0.5d)}$$

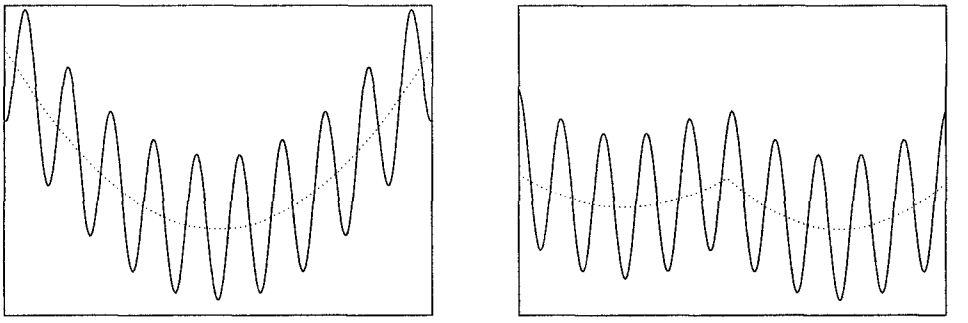


Figure 7.7: Diagonal slices of Rastrigin's function (left) and a double-Rastrigin (right) instance with $d = 3$ and $s = 1$.

7.3 Understanding the Impact of Global Structure

In this section, we explore how the characteristics of the *double-sphere*, which we measure in terms of s and d, impact search. We compare CMA-ES, CHC, and a simple evolution strategy, *Cumulative Step-length Adaptation* (CSA-ES) [73], which is the CMA-ES algorithm without the covariance matrix update. This algorithm is similar to self-adaptation discussed in chapter 2 except that it uses non-local information to adapt the global step-size.

We measure performance in terms of *success rate*, which we denote as ω, and define as the probability that an algorithm will converge to the global optimum. In each experiment, we estimate ω by running 1000 trials of each algorithm and counting the number of instances that find the global optimum.

Our results show that population-based methods are vulnerable to the size of each funnel, as controlled by s, when the depth of the two funnels are relatively close. That is, there exists some funnel characteristics where the exploration process will misguide search into the biggest funnel, not the deepest.

This section is organized in the following way. First, we measure the performance of local search in order to get a rough estimate of the size of each basin of attraction over a range of s and d values. Then we investigate how CMA-ES and CHC perform on the double-sphere. Although this problem only has two local optima, we still find both algorithms can fail even when the size of the optimal basin of attraction is fairly large. Finally, we discuss why this is important from a global optimization perspective by evaluating CMA-ES and CSA-ES on the double-Rastrigin function.

7.3.1 Local Search Properties of the Double-Sphere

As a baseline perspective on the relative size of each basin of attraction, we use the success rate of a local search method, where the probability of finding the global solution is proportional to the size of the basin of attraction to the optimum. However, we don't actually need to search; we simply assign a random point within the domain of the problem to the global solution if:

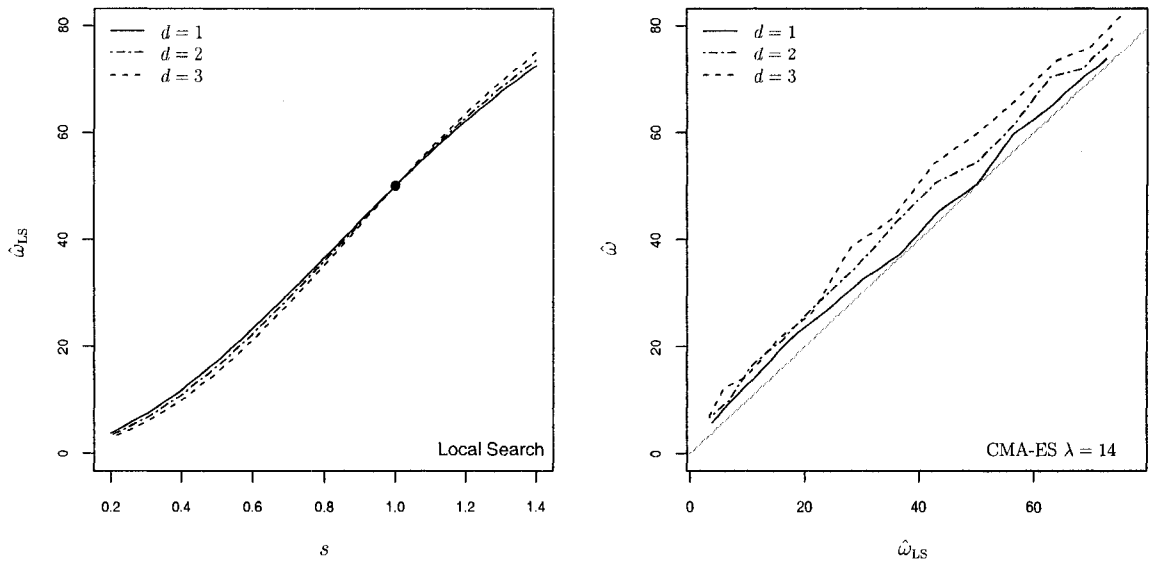
$$\sum_{i=1}^{N}(x_i - \mu_1)^2 \quad < \quad d \cdot N + s \cdot \sum_{i=1}^{N}(x_i - \mu_2)^2$$

We start by considering the double-sphere with dimension $N = 30$. We vary s between $[0.2, 1.4]$ by increments of 0.1, and evaluate different sub-optimal depths of $d = 1, 2$, and 3.

When we estimate $\hat{\omega}$ for local search, we find a positive and approximately linear relationship between $\hat{\omega}_{\text{LS}}$ and s. That is, as we decrease s, we also decrease the probability of finding the global optimum using local search. This makes sense because a small s value increases the size of the sub-optimal funnel, making the optimal funnel proportionally smaller (e.g. the basin of attraction to the optimum is smaller). Figure 7.8(a) shows the relationship between $\hat{\omega}_{\text{LS}}$ and s. This linear relationship changes by a small amount as we vary d. Notice that when $s = 1$, each funnel occupies $\approx 50\%$ of the search space (black dot).

We use this estimate of the size of each basin of attraction as a baseline for interpreting our results. That is, instead of graphing $\hat{\omega}$ for each algorithm as a function of s, we plot the $\hat{\omega}$ values as a function of $\hat{\omega}_{\text{LS}}$, the estimate size of basin of attraction to the global optimum. This makes it easier to observe when the evolutionary search is under- or over-performing with respect to what we would expect from local search.

For example, Figure 7.8(b) shows the success rates of CMA-ES using the default population size of $\lambda = 14$ (for $N = 30$). Since CMA-ES is always above the gray $45\,\text{deg}$ line, we can observe that the success rates for CMA-ES are greater than that of local search. However, there is still a strong *linear* relationship between $\hat{\omega}$ and the size of the optimal funnel. When the depth of the sub-optimal funnel is closer to the optimal ($d = 1$), the success rates for CMA-ES more closely match those of local search.



(a) Local Search

(b) CMA-ES $\lambda = 14$

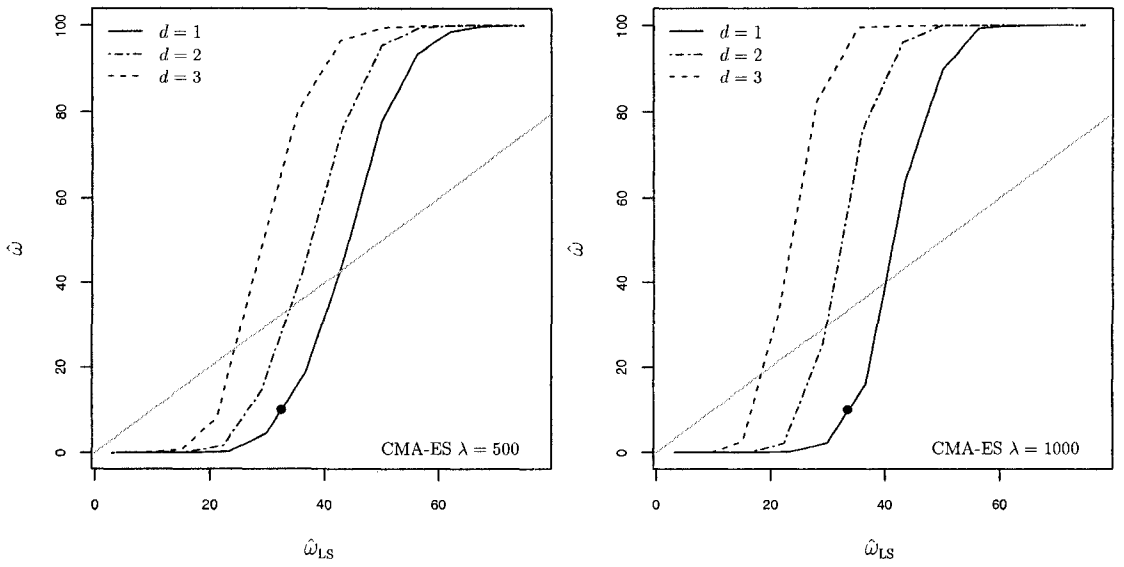
Figure 7.8: Properties of the double-sphere: Figure 7.8(a) shows the success rate of local search as a function of s. There is an approximately linear relationship between the size of the optimal funnel and the success rate for local search, $\hat{\omega}_{\text{LS}}$. Notice that the depth of the sub-optimal funnel does not greatly impact $\hat{\omega}_{\text{LS}}$. Figure 7.8(b) shows the success rate for CMA-ES using the default population size as a function of $\hat{\omega}_{\text{LS}}$. The success rates for CMA-ES are greater than that of local search, but still strongly tied to the size of the optimal funnel ($\approx \hat{\omega}_{\text{LS}}$).

7.3.2 Global Search Properties of the Double-Sphere

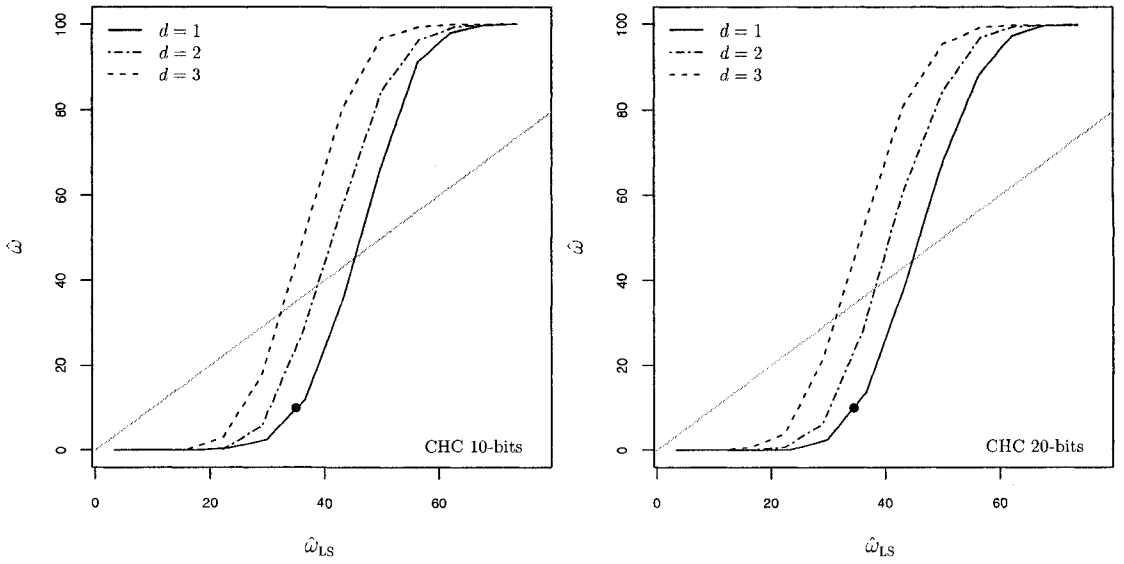
Most evolutionary algorithms perform better on multimodal surfaces when they use a larger population size. This is especially true for CMA-ES [41; 50] and CSA-ES [50]. CHC is probably the exception, as it was designed to use smaller populations [103].

In this section, we would like to understand how the double-sphere impacts global search. In the next section we will consider a range of population sizes for CSA-ES and CMA-ES, but for now, we fix the population size of CMA-ES to $\lambda = 500$ and $\lambda = 1000$. For CHC we use the default of population size of 50 and consider 10- and 20- bits of precision. Our results did not change dramatically with increased population sizes for CHC. A maximum of $100,000$ evaluations were allocated and *no random restarts* were used (expect for the soft-restarts used by CHC). We discuss the role of restarts in the next section.

We observe a similar probability distribution for each algorithm. Instead of a linear trend, as observed for the local search methods in Figures 7.8(a) and 7.8(b), the distribution is bent, or pulled,



(a) CMA-ES with $\lambda = 500$ (left) and $\lambda = 1000$ (right)



(b) CHC with 10-bits (left) and 20-bits (right)

Figure 7.9: CMA-ES and CHC on the double-sphere: The probability of success as a function of the size of the basin of attraction to the optimal funnel, as estimated with local search ($\hat{\omega}_{\text{LS}}$). The gray line indicates the success probability of local search. For each algorithm, the trend is similar; when the optimal funnel is relatively large, the success rates for evolutionary algorithms are high. When the relative size of the optimal funnel is low, evolutionary search is more likely to fail.

into a sigmoid. When the optimal funnel is proportionally larger than the sub-optimal funnel (using local search as an estimate), success rates are extremely high. However, when the optimal funnel is proportionally smaller, the success rates for CMA-ES and CHC drop dramatically. Figures 7.9(a) and 7.9(b) show the probability of success for CMA-ES and CHC as a function of the basin of attraction size for the double-sphere function.

Consider the extreme cases. When the relative size of the optimal funnel is $\approx 70\%$, evolutionary search is highly successful ($\hat{\omega} \approx 100\%$). This means that when local search finds the optimal solution $\approx 70\%$ of the time, CMA-ES and CHC will almost always find the optimal solution. On the other hand, when the relative size of the optimal funnel is only $\approx 10\%$, CMA-ES and CHC fail to find the global optimum. This is true for all the d values we considered.

As we increase d, we increase the relative depth of the sub-optimal and optimal funnel. Figures 7.9(a) and 7.9(b) show that larger values of d shift the $\hat{\omega}$ distribution to the left, meaning that a smaller s value, and therefore, a smaller basin of attraction to the global optimal, is required to observe failure. In other words, the hardest problems for CHC and CMA-ES are those where the depths of the two funnels are close ($d = 1$) and the basin of attraction to the optimal funnel is comparatively small (s is small).

The black dots in Figures 7.8(a) and 7.8(b) represents a success rate of 10% for each algorithm when $d = 1$. This means that CMA-ES will succeed less than 10% of the time even when the relative size of the optimal funnel is $\approx 33\%$ for $\lambda = 500$ and $\lambda = 1000$. A similar problem occurs with CHC. Even when the basin of attraction to the global optima is $\approx 35\%$, the success rate for CHC is about 10%. In general, when local search finds the optimal solution $\approx 1/3$ of the time, the evolutionary algorithms we tested are likely to fail when the depths of the two funnels are relatively close.

The key observation we make in this section is this: when the depths of two funnels are close (e.g. $d = 1$, about 17% different from the barrier that divides them), the global search parameter settings employed by the evolutionary algorithms we tested are likely to cause failure, even when the optimal basin of attraction is relatively large, $\approx 30\%$. As we increase the depth of the sub-optimal funnel, evolutionary search is more successful.

7.3.3 Implications for Global Search: Double-Rastrigin

Considering that CMA-ES using the default population size has probabilities of success that are similar to, or even better than, that of local search, why should we care about the bias of larger populations? The main reason this matters is that if an algorithm cannot cope with the simple structure of the double-sphere, it will also not be successful on more complex multimodal surfaces, like the double-Rastrigin, where the double-sphere dictates the underlying global structure.

We consider three 30-dimensional functions: Rastrigin, double-sphere, and double-Rastrigin. For the double-sphere and the double-Rastrigin, we created instances that are intentionally difficult for CMA-ES by choosing $d = 1$ and $s = 0.7$, which corresponds to an $\hat{\omega}_{\text{LS}} \approx 30\%$. We only consider CSA-ES and CMA-ES because they have strong termination criteria and can solve the 30-dimensional Rastrigin function with large populations. This simplifies the interpretation of our results.

The leftmost graph in Figure 7.10 shows the estimated success rates for the ES algorithms, *without restarts*, on Rastrigin's function as the population varies from $[100, 1000]$ by increments of 100. We have also included the default population size of $\lambda = 14$. These results are consistent with previously reported success rates [41; 50]. The noticeable trend is that larger populations are more able to exploit the underlying sphere structure of the Rastrigin function and locate the best solution. Smaller population sizes tend to get stuck in one of the many local optima. For example, CMA-ES with a population of $\lambda = 14$ never finds the global solution. Using a population size of $\lambda = 100$, CMA-ES only finds the optimal about 10 out of 1000 times.

As we vary the population size for the ES algorithms on the double-sphere, we find the opposite is true. High success rates are realized with low population sizes, but larger values of λ cause CSA-ES and CMA-ES to exhibit extremely low success rates. The right-most graph in Figure 7.10 shows these results.

This presents an interesting trade-off for the double-Rastrigin function: find a population size that balances the difficult characteristics of both the *modality* of the Rastrigin function and the *structure* of the double-sphere. Unfortunately, this balance is disappointing. When we run both algorithms on the double-Rastrigin function, we find that the success rates are lower than 3%, regardless of population size. This is because the success rates for the double-Rastrigin function can

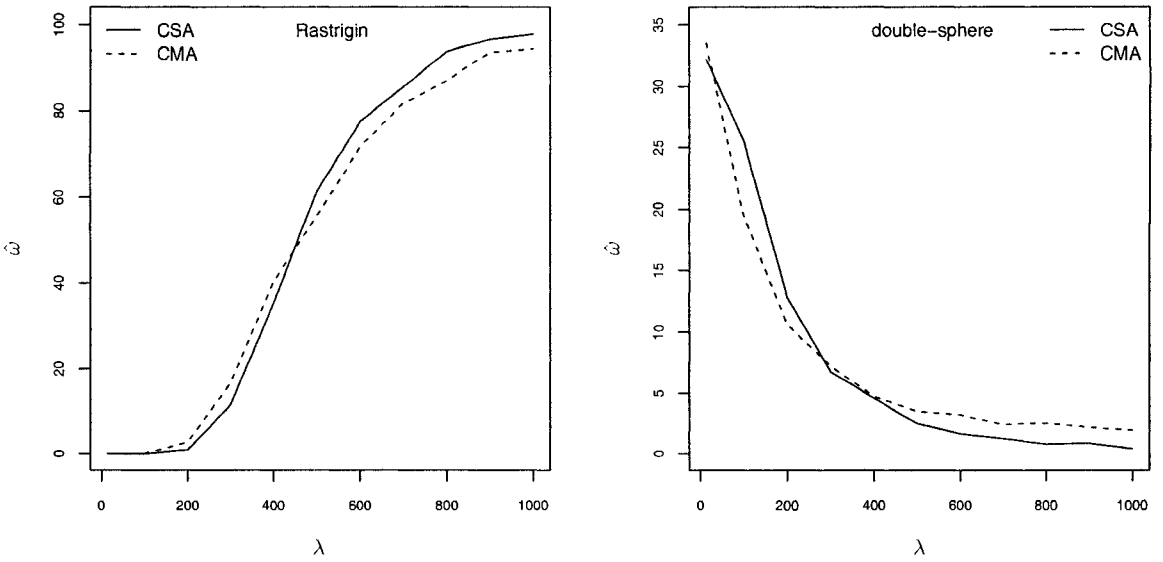


Figure 7.10: Increasing the population size (λ) increases the probability that each evolution strategy will find the optimal solution on Rastrigin's' function (left), but decreases the probability of success on the double-Sphere function.

be decomposed into the success rates of its components. That is, the probability that an algorithm will be successful on the double-Rastrigin is approximately the joint probability that it is successful on the Rastrigin function *and* the probability that it will succeed on the double-sphere.

This is also an incomplete picture because the results presented so far have not used *random restarts*. From a practical point of view, restarts can increase performance because the success probabilities will add. That is, each restart represents an independent event. So, we are not just forced to find a population that balances the characteristics of both the Rastrigin and double-sphere function, we also need to account for the general observation that smaller populations will use fewer evaluations and restart more often.

When we include restarts and allow each algorithm to use $1e7$ evaluations, we still observe low success rates. Figure 7.11 shows the success rates for CSA-ES (left) and CMA-ES (right) on the double-Rastrigin problem as a function of population size with (solid line) and without (dashed line) restarts. The dashed lines in each case show the low probabilities that either algorithm will converge without using restarts (e.g. $\hat{\omega} < 3\%$). With restarts, CSA obtains higher success rates because it is not adapting a covariance matrix and, as a result, is using few evaluations [50].

The main observation we make is that, even if we find a population size that balances trade-

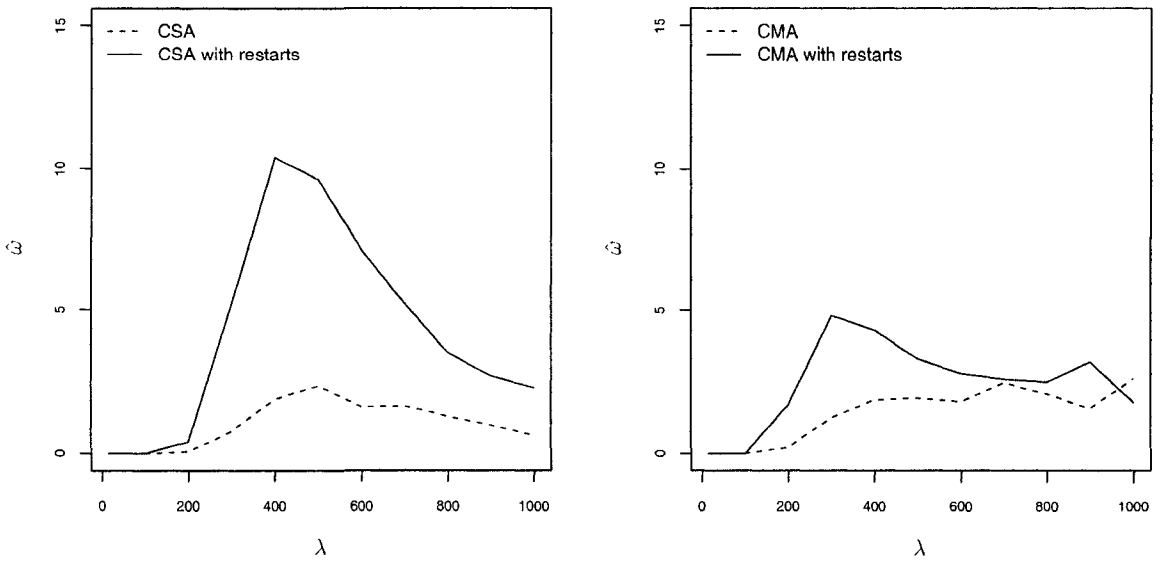


Figure 7.11: The success rates for CSA-ES (left) and CMA-ES (right) with and without restarts on the double-Rastrigin problem as a function of population size λ.

off between modality, global structure, and restarts, we still notice low success rates on a problem where the global basin of attraction is still 30% of the search space. For example, CSA-ES peaks at $\hat{\omega} \approx 11\%$ with a population of $\lambda = 400$. CMA-ES operating with $\lambda = 300$ yields an expected best of $\hat{\omega} \approx 5\%$.

The results of this section reinforce the notion that an algorithm's success or failure largely depends on its ability to cope with the features of a function. A population size suitable for Rastrigin's function is a poor choice for the double-sphere and vise-versa.

There are algorithm configurations that would appear to perform well on both, but that would ultimate perform poorly on the double-Rastrigin. For example, in the previous chapter we discussed IPOPCMA-ES [7], which is a variation of CMA-ES that uses an increased population size after each restart. Since it begins with the default population size, we would expect a success rate on the double-sphere that is proportional to the size of the optimal funnel. Given enough restarts, IPOPCMA-ES would also be successful on the Rastrigin function. However, on the double-Rastrigin function, its performance would be unsatisfactory because no single restart is using a population size that addresses modality and global structure at the same time.

7.4 Limiting Exploration with Dynamic Populations in CSA-ES

Metropolis-type local search algorithms, such as *basin-hopping* and *local optima smoothing*, are effective because they do not explore the entire search space, but rather exploits a single funnel at a time. By quickly comparing the best solution in each funnel, they are more adept to solving multi-funnel problems. This seems to indicate that the best global search methods for multi-funnel problems explore and exploit on a local, not global, level.

On the double-sphere function, larger populations in CSA-ES (and CMA-ES) tend to pull the mean towards the funnel with the most samples. When the funnels are close in depth, a larger sub-optimal funnel is more likely to have more samples. Smaller populations are less vulnerable to this because less information being sampled. On the double-Rastrigin, we need the best of both worlds: a small population size to drop into a funnel without being pulled towards a larger basin of attraction, and then a large population size to exploit the underlying structure of that particular funnel.

As a proof of concept, we implemented CSA-ES with a dynamic population size that increased as the global step-size decreased. A decrease in step-size indicates a higher level of exploitation. When search is first exploring, it is utilizing a small population size. As it begins to exploit a promising region, increasing the population size will help exploit the underlying funnel structure. The algorithm is identical to CSA-ES in every way except at the end of each generation, we compute a new population size based on a function of the global step-size σ, the initial step-size σ_0, and an upper bound of the population size, λ_{MAX}.

$$\lambda = \lambda_{\text{MAX}} \left(\frac{\sigma}{\sigma_0} - 1 \right)^2$$

We ensure that λ never falls below the default population size, $\lambda_d = 14$, or exceeds the maximum λ_{MAX}, which is an input parameter.

We ran this strategy, which we denote D-CSA-ES, on the 30-dimensional Rastrigin, double-sphere, and double-Rastrigin functions for the same values of λ used in the previous section, except that D-CSA-ES interprets this value as λ_{MAX}. The resulting search strategy is less effective on the Rastrigin function, but operates at a consistent level on the double-sphere function that is proportional to the size of the optimal funnel, regardless of the population size. The left graph in Figure

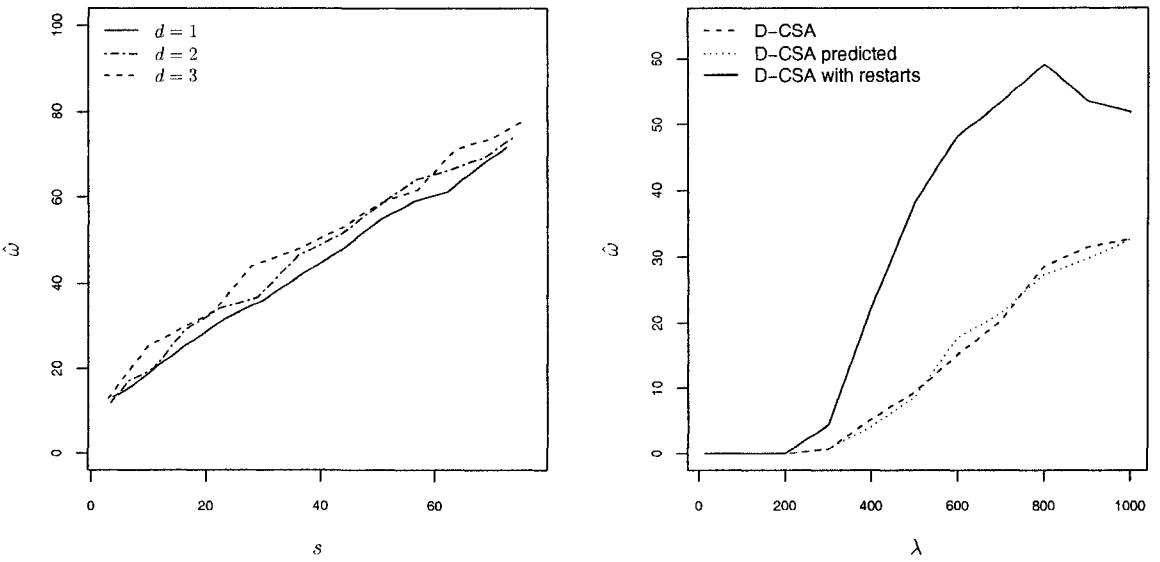


Figure 7.12: D-CSA-ES on the double-sphere (left) and on the double-Rastrigin (right). The relationship between success rate and the size of the optimal funnel remains linear. This results in a much higher success rate on the double-Rastrigin function.

7.12 shows D-CSA-ES on the double-sphere as a function of optimal funnel size for $d = 1, 2$, and 3. The most striking feature is the approximately linear relationship between the size of the optimal funnel and the success rate of D-CSA-ES. This resembles the relationship of CMA-ES using a default population size on the double-sphere, but with a setting for λ that is more appropriate for global optimization.

This figure also highlights one drawback to this method; the effectiveness of exploiting one funnel at a time, as measured in terms of success rate, is proportional to the size of the optimal funnel. This means that on double-sphere function instances, where the optimal funnel is proportionally larger than the sub-optimal funnel, algorithms like CMA-ES and CHC will achieve higher success rates than D-CSA-ES.

What does this mean for the double-Rastrigin function? The right graph in Figure 7.12 show D-CSA-ES on the double-Rastrigin function for $s = 0.7$ and $d = 1$ as a function of population size. Without restarts (dash), D-CSA-ES has a success rate the is about 10 times higher than either CMA-ES or CSA-ES. When D-CSA-ES runs with restarts (solid line) until $1e7$ evaluations, it success rates are as high as $\approx 60\%$.

The dotted line in this graph represents the predicted performance obtained by multiplying the $\hat{\omega}$

from Rastrigin with $\hat{\omega}$ from the double-sphere. The prediction is very close to the empirical results and reinforces the notion that successful search must cope with both modality and global structure.

7.5 Summary

Global structure can clearly impact the performance of evolutionary optimization. When the optimal funnel is proportionally smaller, the success rates for CHC and CMA-ES decrease dramatically on the double-sphere, especially when the depths of the two funnels are close. Exploration is not able to distinguish between funnel quality, and is pulled into the larger funnel. We believe these results generalize to other algorithms.

This presents a problem for CMA-ES and CSA-ES on the double-Rastrigin function because, although larger population sizes are necessary to exploit the underlying structure of the Rastrigin, they are also more bias towards funnel size. The population size that is best for Rastrigin is the least effective on the double-sphere. A compromise that works on both is disappointing.

By dynamically adapting the population size, D-CSA-ES is less biased toward funnel size while exploring the search space. However, as it descends into a particular funnel, and it begins to exploit the search space, increasing the population size aids D-CSA-ES in detecting the underlying structure of the funnel and avoiding local optima. This results in a strategy whose success rate is dependent on funnel size; when the optimal funnel is large, the success rates for D-CSA-ES are not a good as CHC or CMA-ES. But when the optimal funnel is small, D-CSA-ES will still find the global solution with a probability proportional to relative funnel size. The highs are not as high, but the lows are still acceptable.

This is the first step toward understanding how global structure impacts search. Future work should implement a dynamic population size in CMA-ES, and see if that is effective with the more complex distribution used in CMA-ES. This may result in a search method capable of finding the best solutions to the Lennard-Jones instances where the optimum is hidden in a funnel that is relatively small when compared to the search space.

Exploring the search space to gain a global perspective before exploiting a particular region may be an effective strategy for "big valley", single-funnel problems. But on multi-funnel landscapes, the effectiveness of exploration comes into question as a global search strategy. This work supports

an ongoing awareness that, if an algorithm is going to be successful, then it must be able to deal with the features in the landscape.

Chapter 8

Conclusion

This research is primarily concerned with evolutionary parameter optimization and its relationship with *ridges* and *global structure*. In the process of understanding how these features impact search, we have applied several algorithms from different domains on a variety of test functions and realistic applications.

The algorithm selection used here is broad. When appropriate, we have compared the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and the CHC genetic algorithm with direct-search algorithms, like Generalized Pattern Search (GPS) and Mesh Adaptive Direct Search (MADS), and local search, as well as with gradient-based methods, such as the BFGS quasi-Newton method and the conjugate-gradient method. In some instances, we have compared evolutionary search with specialized algorithms such as the Levenberg-Marquardt, Salomon's optimize and refine, and basin hopping. We also included two new algorithms: tube search and a variable population size Cumulative Step-length Adaptation Evolution Strategy (CSA-ES). This relatively large base of strategies increases the confidence we have in our results because we are more certain that we are comparing the most effective and efficient strategies.

The test functions considered were also diverse. We rotated the standard set of benchmark problems in order to increase the degree to which the parameters interact. We also introduced some new test functions. In chapter 3, we presented two modified Griewangk functions, where the difficult scales with problem size, and also describe a realistic version of the "static corrections" problem from geophysics. Chapter 7 introduces a class of double-funnel landscapes.

Real applications have also played a vital role. Motivation for studying ridges and global structure is partly due to the temperature retrieval problem and difficulty Lennard-Jones cluster instances.

These real applications add meaning and integrity to our conclusions. In fact, using these real applications as a basis for studying ridges and global structure is a significant contribution in this dissertation.

When we compare this diverse selection of algorithms on these difficult test problems, we discover that no single algorithm is best. This result is widely accepted within the optimization community. For example, Lewis *et al.* reason that, "... since nonlinear optimization problems come in all forms, there is no one-size-fits-all algorithm that can successfully solve all problems" [54]. Our research extends this notion by observing that algorithm differences can be directly related to problem features.

We have observed that ridges and global structure are two features that have largely been ignored by much of the evolutionary algorithm community. Yet these features clearly have an impact on search. We conclude our results from two perspectives. First, we recap how these two features impact some of the general-purpose evolutionary algorithms we have discussed. Then we summarize how the most efficient and effective methods find competitive solutions on applications and test functions where ridges and certain types of global structure exist. We close this chapter with a summary of implications.

8.1 Evolutionary Search and Feature Interaction

We revisit what we have learned about how evolutionary parameter optimization interacts with ridges and global structure.

8.1.1 Ridges

One of the main conclusions we observe with respect to ridge functions is that, unless an algorithm's heuristics intentionally addresses the ridge problem, it is likely to behave inefficiently or fail on ridge problems. This is not unique to evolutionary search. Simple methods from other domains, like steepest-descent and local search, are often also less efficient, and sometimes fail, on ridge surfaces. Algorithms that do address the ridge problem seem to incur additional complexity in order to gain overall efficiency. For example, both CMA-ES and quasi-Newton methods are more complex, but overall, these algorithms are more robust, efficient, and effective on ridge functions.

Algorithms that fail do so in different ways. A bias search direction, such as those used by local search and GPS, will behave inefficiently or fail on ridges when their search direction is not aligned with the ridge axis. But an unbiased search direction is not the solution. Algorithms like the Self-Adaptive Evolution Strategy and MADS are also inefficient on ridges.

The local search described in this paper creates a fixed neighborhood of search directions by changing only one variable at a time. This representation creates false local optima on ridges that are not aligned with the coordinate axis. Increasing the precision generally increases the effectiveness, but comes with a high cost: Adding 1-bit of precision increase the number of steps search will take by n (number of dimensions). Changing the search direction helps increase efficiency, but getting the right direction adds complexity and is difficult as the number of parameters increase.

Self-adaptation samples points in an unbiased way. On smooth ridge functions, self-adaptation will continue to decrease its step-size until it removes the bias toward selecting smaller step-size values. On the sharp ridge function, this inherent bias cannot be removed, and the step-size will continually decrease until the strategy fails. This explains why the performance of self-adaptation is poor on ridge functions.

The current impression of self-adaptation within the evolutionary optimization community is this: given the shortcomings of *correlated mutations* and the lack of robustness when adapting *individual step-sizes*, the only safe way to self-adapt is with a single strategy parameter—the global step-size σ. But on ridge functions, self-adaptation creates a σ that is too small, causing this strategy to behave inefficiently. All of these remarks seem to indicate that the self-adaptation, on any level, does not work.

Our research seems to support the opposite; self-adaptation works as expected. Using a small step-size, the selected individuals *will tend to have* distributions that are more likely to explore better regions of the search space on the ridge function. A large step-size is less likely to "explore" better regions of the search space because of the inherent bias that comes with ridge functions.

Successful algorithms on ridge functions use a rotated representation. However, a Principal Components Analysis on a localized sample of best points can create a direction of maximum variance that is orthogonal to the gradient. This occurs when the neighborhood around the sample is approximately linear. In this instance, the direction created by PCA is a poor choice. Figure 8.1

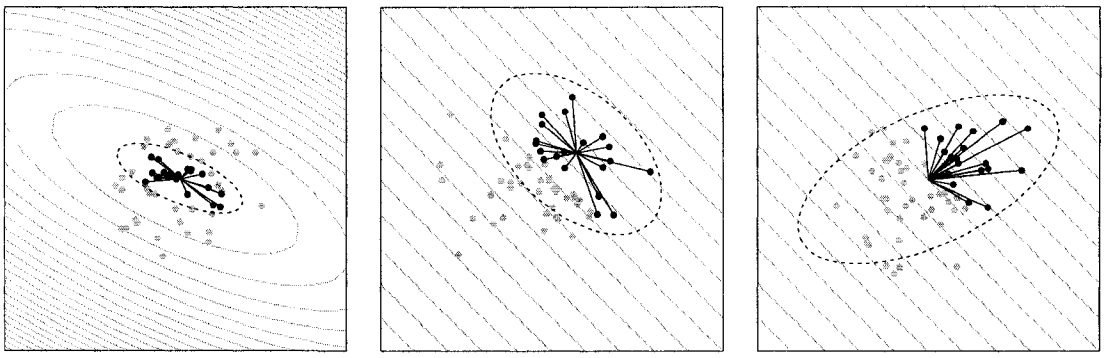


Figure 8.1: Learning rotations with PCA: The left-most graph shows a distribution (dashed line) defined by using a Principal Components Analysis on the best points (black dots) of an isotropic sample (black and gray dots). When the localized sample is on the ridge, using a Principal Components Analysis on the best points is an effective method for computing a rotation. However, when the localized sample is not on the ridge (middle figure), PCA can create a direction of maximum variance orthogonal to the gradient. Here the distribution (dashed line) is shrinking in a direction that is less effective. This is why CMA-ES uses the "steps" to the best points and not the points themselves (right-most figure). Here the distribution (dashed line) is computed using the mean of the sample, not the mean of the best points, as is the case in the middle graph.

shows a simple example of this problem. This is why CMA-ES uses the "steps" to the best points and not the best points themselves when using a "rank-μ" update. If path information is used, the rotation acts as a kind of "momentum" term that keeps the search moving in its current direction. In effect, PCA exploits information about variance, whereas methods like Gram-Schmidt uses path information.

From a practical point of view, our understanding of ridges explains why many well-known evolutionary algorithms and local search methods perform poorly on the temperature retrieval problem. We know there are ridges in the search space and that most algorithms cannot move efficiently along this feature. The rotated representation used by CMA-ES overcomes many of the problems associated with adapting rotational parameters and has a clear advantage on the temperature problem.

8.1.2 Global Structure

Our dispersion metric distinguishes functions based on their *global structure*. It is a more concrete way of describing the underlying structure of a problem than other vague descriptions. Dispersion also does not have the unrealistic requirement that the best local optima of the search space be enumerated, as is the case with disconnectivity graphs, barrier trees, and Locatelli's method. On low

dispersion functions, the best regions of the search space become more localized as the threshold decreases. The opposite is true for high dispersions functions; as the threshold decreases, the best regions of the search space tend to become more disperse.

Mobility measures the quality and dispersion of the local optima visited during search. Defining a threshold serves two purposes: It creates a basin of attraction that is close to local optima in terms of fitness, and it focuses our research on only the best local optima. We found that algorithms with higher mobility, such as CHC, tend to have more effective solutions in low dimensions. This trend is difficult to capture in higher dimensional space (e.g. greater than ten).

Our dispersion metric predicts how efficient and effective CMA-ES will be on a given function. We find that on *low dispersion* functions, CMA-ES performs as expected because the underlying structure is convex, like the sphere function. We have identified two reasons why CMA-ES is less efficient on *high dispersion* functions. The most serious issue is that the step size adaptation mechanism used by CMA-ES, called *cumulation*, does not work as expected when the best regions of a function are disperse. The primary force that decreases the offspring distribution comes from the slow collapse of the covariance matrix. We also noticed that CMA-ES can waste up to 20% of its evaluations simply finding the boundaries of the problem on high dispersion functions. It is not clear how to address this constraint handling problem.

One of the problems with the high dispersion benchmark test functions is that the number of funnels increase with dimensionality and there is no way to control the size of each funnel. The new class of double-funnel test functions presented in chapter 7 keeps the number of funnels constant (two) as the problem size increases, and also allows the relative characteristics of the sub-optimal funnel, such as depth and size, to change.

We use several instances of these double-funnel test problems to challenge the heuristic belief that exploring the search space first, to gain a global perspective, is an effective way to identify the best regions of the search space. We find that when the global optimum is located in a funnel that is proportionally smaller in the search space, and sub-optimal funnel is relatively close in objective function value, the success rates for CHC and CMA-ES are extremely low.

We observe that exploring the search space first, before exploiting a particular region may be an effective strategy on some problems, especially those with a single-funnel or big valley topology.

But our results suggest that in high dimensions, the effectiveness of this philosophy comes into question as a general purpose global optimization technique.

8.2 Specialized Algorithms

Throughout this dissertation, we have showed that algorithms exploiting problem-specific knowledge tend to find more effective and efficient solutions. Here we summarize the specialized algorithms that performed well on ridge surfaces and problems that exhibit a certain type of global structure.

8.2.1 Ridges: Searching for Temperature Profiles

On the temperature problem, Salomon's *optimize and refine* method and our *tube search* strategy produce approximate solutions quickly by exploiting the physical continuity of the expected temperature profiles. One drawback here is that the temperature profiles found are imprecise. If we restrict ourselves to direct search methods, two choices emerge; either retrieve a quick and dirty solution using a strategy that exploits continuity or spend more time finding an effective temperature profile using an evolutionary algorithm like CMA-ES.

The temperature problem we tested has a gradient and the objective function is modeled as a nonlinear least-squares function. We know that when gradient information is available, the using it will likely result in a more efficient and effective search, especially on low-modality problems where restarts are a reasonable global search strategy. The performances of the BFGS and conjugate-gradient algorithm were unsatisfying. On the one hand, the Levenberg-Marquardt algorithm was incredibly fast. The difference is that the Levenberg-Marquardt exploits the Hessian structure of the objective function, whereas the other gradient-based methods do not. On temperature retrieval problems, where derivative information is available and the objective function can be stated as a least-squares problem, the Levenberg-Marquardt is clearly the best choice.

8.2.2 Global Structure: Funnel Characteristics

The basin hopping strategy is effective because it does not explore the entire search space, but rather exploits a single funnel at a time. It quickly descends to competitive regions with in each funnel

as a way of comparing multiple funnels. This indicates that exploiting on a local level is actually a better global search strategy on multi-funnel test functions.

When we dynamically adjust the population size of CSA-ES, we limit its ability to explore. Counter-intuitively, we find that this is a much more effective global search method on some of the difficult double-funnel test functions. Limiting the amount of exploration actually improves its global search performance. This is because the probability of success for D-CSA-ES on a double-funnel landscape is proportional to the size of the optimal funnel. Multiple restarts increase its probability of find the optimal solution because the overall probability is the sum of each independent search.

Comparing funnels has some limitations. The effectiveness of exploring funnels is limited to problems that contain relatively few funnels. That is, algorithms such as basin hopping and D-CSA-ES must rely on several *restarts* in order to compare the best solutions that exist within each funnel.

8.3 Implications

The evolutionary optimization and direct search communities have still not developed a clear understanding of how different algorithms exploit different types of problem features. And while most researchers agree that every algorithm has a special niche where it will perform the best, an algorithm's strengths and weaknesses are rarely reported, especially how they relate to problem features.

Historically, it seems that evolutionary search was concerned primarily with modality. This makes sense because gradient-based local search algorithms get stuck in local optima and using restarts as a global search strategy is often not effective. The evolutionary search community found a niche and developed population-based algorithms that seemed to avoid getting trapped in local optima and find more globally competitive solutions on multimodal surfaces. It wasn't until the mid 1990's that researchers really started to realize that separability was also a concern.

This work brings light to the fact that ridges and global structure are also features that impact search. Like modality and separability, we should also pay attention to these features and implement strategies that directly address the difficulties they pose. This will likely change how new heuristics are created and result in more diversity among algorithms. As algorithms become more specialized

in one domain, we should see a greater variance in overall performance.

Inspiration for new ideas in search still frequently come from a natural metaphor. But we have seen that algorithms that address ridges or global structure are very intentional with respect to these features. For example, CMA-ES was created with the scale and separability in mind. The name "basin hopping" suggests that this algorithm was specifically designed for problems that have funnels. As new methods emerge, the overall role that evolutionary algorithms play in the global optimization community will expand. In the process, we will learn more about how certain features, that make optimization difficult for search, interact with our heuristic ideas.

REFERENCES

[1] D. H. Ackley. *A connectionist machine for Genetic Hillclimbing*. Kluwer Academic Publishers, 1987.

[2] B. Addis, M. Locatelli, and F. Schoen. Local optima smoothing for global optimization. 2003.

[3] D. Arnold and H.-G. Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24(1):135–159, 2003.

[4] D. Arnold and H.-G. Beyer. Performance Analysis of Evolutionary Optimization with Cumulative Step Length Adaptation. *IEEE Transactions on Automatic Control*, 49(4):617–622, April 2004.

[5] D. V. Arnold and H.-G. Beyer. Evolution Strategies with Cumulative Step-length Adaptation on the Noisy Parabolic Ridge, 2006.

[6] C. Audet and J. J. Dennis. Mesh adaptive direct search algorithms for constrained optimization. CAAM Technical Report TR04-02, Rice University, Texas., 2004., 2004.

[7] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *Proceedings of IEEE Congress of Evolutionary Computation*, 2005.

[8] T. Back. *Evolutonary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[9] T. Bäck. Personal communication, 2006.

[10] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1, 1993.

[11] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann Publishers.

[12] U. Bartling and H. Mühlenbein. Optimization of large scale parcel distribution systems by the breeder genetic algorithm (bga). In *ICGA*, pages 473–480, 1997.

[13] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation*. Springer, 2006.

[14] H.-G. Beyer. On the Performance of $(1, \lambda)$-Evolution Strategies for the Ridge Function Class. *IEEE Transactions on Evolutionary Computation*, 5(3):218–235, 2001.

[15] H.-G. Beyer and H.-P. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, May 2002.

[16] K. Boese, A. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16, 1994.

[17] I. O. Bohachevsky, M. E. Johnson, and M. L. Stein. Generalized Simulated Annealing for Function Optimization. *Technometrics*, 28:209–217, 1986.

[18] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimal conditions. *Journal of the Royal Statistical Society*, 13, 1951.

[19] R. P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, New Jersey, 2002.

[20] Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.

[21] G. Couture, C. Audet, J. Dennis, and M. Abramson. Nomad software package, 2007.

[22] W. Davidon. Variable Metric method for minimization. *SIAM Journal Optimization*, 1:1–17, 1959.

[23] L. Davis. Bit-Climbing, Representational Bias, and Test Suite Design. pages 18–23. Morgan Kaufmann, 1991.

[24] K. DeJong. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. *PhD thesis, University of Michigan, Ann Arbor*, 1975.

[25] J. Doye, R. Leary, M. Locatelli, and F. Schoen. Global Optimization of Morse Clusters by Potential Energy Transforms. *INFORMS Journal on Computing*, 16(4):371–379, 2004.

[26] J. Doye, M. Miller, and D. Wales. Evolution of the Potential Energy Surface with Size for Lennard-Jones Clusters. *Journal of Chemical Physics*, 111:8417, 1999.

[27] J. P. Doye. Physical perspectives on the global optimization of atomic clusters. In *Chapter in forthcoming Kluwer book Global optimization – select case studies*, 2006.

[28] J. P. Doye, M. A. Miller, and D. J. Wales. The double-funnel energy landscape of the 38-atom Lennard-Jones cluster. *Journal of Chemical Physics*, 110(14), April 1999.

[29] A. E. Eiben and T. Back. Empirical Investigation of Multiparent Recombination Operators in Evolution Strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[30] R. Englen, A. Denning, K. Gurney, and G. Stephens. Global Observations of the Carbon Budget: I. Expected Satellite Capabilities for Emission Spectroscopy in the EOS and NPOESS Eras. *Journal of Geophysical Research*, 106:20,055–20,068, 2001.

[31] L. Eshelman and D. Schaffer. Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms*, 1993.

[32] L. J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *FOGA*, 1991.

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

[34] Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.

[35] R. Fletcher and C. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:194–154, 1964.

[36] L. Fogel, A. Owens, and M. Walsh. *Artificial Intellegence Through Simulated Evolution.* John Wiley, 1966.

[37] Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

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

[39] N. Hansen. Invariance, Self-adaptation and Correlated Mutations in Evolution Strategies. In *Proceedings of Parallel Problem Solving from Nature*, pages 355–364, 2000.

[40] N. Hansen. An Analysis of Mutative σ-Self-Adaptation on Linear Fitness Functions. *Evolutionary Computation*, Accepted.

[41] N. Hansen and S. Kern. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *PPSN*. Springer, 2004.

[42] N. Hansen, S. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[43] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[44] M. Herdy. Reproductive Isolation as Strategy Parameter in Hierarchically Organized Evolution Strategies. In *Proceedings of Parallel Problem Solving from Nature*, pages 207–217, 1992.

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

[46] R. Hooke and T. Veeves. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8:212–229, 1961.

[47] K. hsin Liang, X. Yao, and C. Newton. Adapting Self-adaptive Parameters in Evolutionary Algorithms. *Applied Intelligence*, 15:171–180, 2001.

[48] T. C. Hu, V. Klee, and D. Larman. Optimization of globally convex functions. *SIAM Journal on Control and Optimization*, 27(5), 1989.

[49] S. Kazadi. Conjugate schema and basis representation of crossover and mutation. *Evolutionary Computation*, 6(2), 1998.

[50] S. Kern, S. Muller, N. Hansen, D. Buche, J. Ocenasek, and P. Koumoustakos. Learning Probability Distributions in Continous Evolutinary Algorithms—a Comparative Review. *Natural Computing*, 3:77–112, 2004.

[51] R. H. Leary. Global Optimization on Funneling Landscapes. *Journal of Global Optimization*, 18, 2000.

[52] R. H. Leary and J. P. K. Doye. Tetrahedral global minimum for the 98-atom lennard-jones cluster. *Phys. Rev. E*, 60(6):R6320–R6322, Dec 1999.

[53] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[54] R. Lewis, V. Torczon, and M. Trosset. Direct Search Methods: Then and Now. *Journal of Computational and Applied Mathematics*, 124, 2000.

[55] M. Locatelli. On the multilevel structure of global optimization problems. *Computational Optimization and Applications*, 30, 2005.

[56] M. Lunacek and D. Whitley. Function dispersion and the CMA Evolution Strategy. In *Proceedings of GECCO*, 2006.

[57] M. Lunacek and D. Whitley. Searching for Balance: Understanding the Behavior of Self-adaptation on Ridge Functions. In *Proceedings of PPSN*, 2006.

[58] M. Lunacek, D. Whitley, P. Gabriel, and G. Stephens. Applying Search Algorithms to the Temperature Inversion Problem. In *Proceedings of GECCO*, 2004.

[59] M. Lunacek, D. Whitley, and J. Knight. Measuring Mobility and the Performance of Global Search Algorithms. In *Proceedings of GECCO*, 2005.

[60] M. Lunacek, D. Whitley, and A. Sutton. The impact of global structure on search. In *Submitted*, 2008.

[61] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal of Applied Mathematics*, 11:431–441, 1963.

[62] K. Mathias, J. D. Schaffer, L. J. Eschelman, and M. Mani. The Effects of Control Parameters and Restarts on Search Stagnation in Evolutionary Programming. In *PPSN*, 1998.

[63] K. Mathias, D. Whitley, T. Kusuma, and C. Stork. An Empirical Evaluation of Genetic Algorithms on Noisy Objective Functions. In S. K. Pal, editor, *Genetic Algorithms for Pattern Recognition*, pages 65–86. CRC Press, 1996.

[64] R. McClatchey, R. Senn, J. Feldy, S. Voltz, and J. Garing. Optical properties of the atmosphere. Technical Report TR-354, AFCRL, 1971.

[65] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. *Journal of Chemical Physics*, 21:1087–1953, 1953.

[66] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive Models for the Breeder Genetic Algorithm. *Evolutionary Computation*, 1(1):25–49, 1993.

[67] H. Mühlenbein, M. Shomisch, and J. Born. The parallel genetic algorithm in combinatorial optimization. *Parrallel Computing*, 7:65–85, 1988.

[68] S. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.

[69] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[70] K. Ohkura, Y. Matsumura, and K. Ueda. Robust Evolution Strategies. *Applied Intelligence*, 15:153–169, 2001.

[71] I. Ono and S. Kobayashi. A real-coded Genetic algorithm for Function Optimization Using Unimodal Normal Distributioin Crossover. In *International Conference on Genetic Algorithms*, pages 246–253, 1997.

[72] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994.

[73] A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-Size Adaptation Based on Non-local use of Selection Information. In *PPSN*, pages 189–198. Springer, 1994.

[74] A. I. Oyman, H. Beyer, and H. Schwefel. Where Elitists Start Limping Evolution Strategies at Ridge Functions. In A. E. Eiben, T. B'ack, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 34–43, Berlin, 1998. Springer.

[75] A. I. Oyman and H.-G. Beyer. Analysis of the $(\mu/\mu, \lambda)$-ES on the Parabolic Ridge. *Evolutionary Computation*, 8(3):267–289, 2000.

[76] A. I. Oyman, H.-G. Beyer, and H.-P. Schwefel. Analysis of the $(1, \lambda)$-ES on the Parabolic Ridge. *Evolutionary Computation*, 8(3):249–265, 2000.

[77] P. M. Pardalos and F. Schoen. Recent Advances and Trends in Global Optimization: Deterministic and Stochastic Methods. In *CAPD*, 2004.

[78] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. Technical Report 98013, Urbana, IL, 1998.

[79] M. Powell. An efficient Method for Finding the Minimum of a Function of Several Variable Without Calculating Derivatives. *The Computer Journal*, 7:155–162, 1964.

[80] L. Rastrigin. Extremal control systems. In *Theoretical Foundations of Engineering Cybernetics Series*, 1974.

[81] H. Rosenbrock. An Automatic Method for Finding the Greatest or Least value of a Function. *Computer Journal*, 3:175–184, 1960.

[82] G. Rudolph. On correlated mutations in evolution strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, pages 105–114, Amsterdam, 1992. Elsevier.

[83] R. Salomon. Re-evaluating Genetic Algorithm Performance Under Coordinate Rotation of Benchmark Functions: A Survey of Some Theoretical and Practical Aspects of Genetic Algorithms. *BioSystems*, 39:263–278, 1996.

[84] R. Salomon. Applying Evolutionary Algorithms to Real-World-Inspired Problems with Physical Smoothness Constraints. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 921–928, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[85] R. Salomon. The curse of High Dimensional Search Spaces: Observing Premature Convergence in Unimodal Funcitons. In *Proceedings of the Congress on Evolutionary Computation*, pages 918–923. IEEE Press, 2004.

[86] D. Schaffer, R. Caruana, L. Eschelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Third international conference on Genetic algorithms*, 1989.

[87] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. *Artificial Intelligence*, 132(2):121–150, 2001.

[88] H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, Inc., 1995.

[89] Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656, 1970.

[90] J. R. Shewchuk. An introduction to conjugate gradient method without the agonizing pain. 1994.

[91] A. Sokolov, D. Whitley, and M. Lunacek. Alternative Implementations of the Griewank Function. In *Preceedings of GECCO*, 2005.

[92] W. Spendly, G. Hext, and R. Himsworth. Sequential Application of Simplex Designs in Optimization and Evolutionary Operation. *Technometrics*, 4:441–461, 1962.

[93] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.

[94] G. Syswerda. Simulated Crossover in Genetic Algorithms. pages 239–255. Morgan Kaufmann, 1993.

[95] V. J. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Houston, TX, USA, 1989.

[96] V. J. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optimization*, 7:1–25, 1993.

[97] V. J. Torczon and M. Trosset. From evolutionary operation to parallel direct search: Pattern search algorithms for numerical optimization. *Computing Science and Statistics*, 29:396–401, 1998.

[98] H.-M. Voigt. On some difficulties in local evolutionary search. In *IEEE*, 1999.

[99] D. J. Wales. Energy Landscapes and Properties of Biomolecules. *Physical Biology*, 2005.

[100] D. J. Wales and J. P. Doye. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *Journal of Chemical Physics*, 101(28), April 1997.

[101] D. J. Wales and H. A. Scheraga. Global optimization of clusters, crystals, and biomolecules. *Science*, 285(27), August 1999.

[102] D. Whitley, L. Barbulescu, and J. Watson. Local Search and High Precision Gray Codes. In *Foundations of Genetic Algorithms FOGA-6*. Morgan Kaufmann, 2001.

[103] D. Whitley, R. Beveridge, C. Graves, and K. Mathias. Test Driving Three 1995 Genetic Algorithms: New Test Functions and Geometric Matching. *Journal of Heuristics*, 1995.

[104] D. Whitley, K. Bush, and J. Rowe. Subthreshold-seeking behavior and robust local search. In *Proceedings of GECCO*, 2004.

[105] D. Whitley, M. Lunacek, and J. Knight. Ruffled by Ridges: How Evolutionary Algorithms can fail. In *Proceedings of GECCO*, 2004.

[106] D. Whitley, M. Lunacek, and A. Sokolov. Comparing the Niches of CMA-ES, CHC and Pattern Search Using Diverse Benchmarks. In *Submitted*, 2006.

[107] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 239–246, San Francisco, CA, 1995. Morgan Kaufmann.

[108] D. Whitley and S. Rana. Representation, search, and genetic algorithms. In *14th National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[109] D. Whitley, S. B. Rana, J. Dzubera, and K. E. Mathias. Evaluating Evolutionary Algorithms. *Artificial Intelligence*, 85(1-2):245–276, 1996.

[110] A. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, 1991.

[111] D. Wyatt and H. Lipson. Finding building blocks through eigenstructure adaptation. In *GECCO*, 2003.

[112] W. Zangwill. Minimizing a Functon without Calculating Derivatives. *The Computer Journal*, 10:293–296, 1967.