NO FREE LUNCH, BAYESIAN INFERENCE, AND UTILITY: A DECISION-THEORETIC APPROACH TO OPTIMIZATION

by

Christopher Kenneth Monson

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science
Brigham Young University
August 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Christopher Kenneth Monson

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Kevin D. Seppi, Chair
Date	Bryan S. Morse
Date	Dan A. Ventura
Date	Michael D. Jones
Date	Irene Langkilde-Geary

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Christopher Kenneth Monson in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Kevin D. Seppi Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

NO FREE LUNCH, BAYESIAN INFERENCE, AND UTILITY: A DECISION-THEORETIC APPROACH TO OPTIMIZATION

Christopher Kenneth Monson
Department of Computer Science
Doctor of Philosophy

Existing approaches to continuous optimization are essentially mechanisms for deciding which locations should be sampled in order to obtain information about a target function's global optimum. These methods, while often effective in particular domains, generally base their decisions on heuristics developed in consideration of ill-defined desiderata rather than on explicitly defined goals or models of the available information that may be used to achieve them.

The problem of numerical optimization is essentially one of deciding what information to gather, then using that information to infer the location of the global optimum. That being the case, it makes sense to model the problem using the language of decision theory and Bayesian inference. The contribution of this work is precisely such a model of the optimization problem, a model that explicitly describes information relationships, admits clear expression of the target function class as dictated by No Free Lunch, and makes rational and mathematically prin-

cipled use of utility and cost. The result is an algorithm that displays surprisingly sophisticated behavior when supplied with simple and straightforward declarations of the function class and the utilities and costs of sampling.

In short, this work intimates that continuous optimization is equivalent to statistical inference and decision theory, and the result of viewing the problem in this way has concrete theoretical and practical benefits.

ACKNOWLEDGMENTS

Some of the most enjoyable years of my life have been spent as a student in the CS graduate program at BYU. I owe a debt of gratitude to many people, both in and out of the department.

My heartfelt appreciation goes to the secretaries and supporting staff for their sincere and capable efforts on my behalf, especially in cases where subtle but important details have been waiting to catch me unawares. Without their help I probably would have embarrassingly misspelled the name of a faculty member, failed to complete sufficient paperwork for employment, sent a fax to the wrong conference, had loans go into repayment early, or been uninsured for a semester.

Naturally, Kevin Seppi deserves a great deal of the gratitude as my adviser, coauthor, and friend. His sincere and caring approach to my education has not only made it possible to obtain this degree, but has also been a catalyst for positive change in my personal life. His academic insights, encouragement in the face of adversity, and unwavering belief in my ability to accomplish things that I had never considered myself capable of achieving have been critical to my success.

Without the support of my wife CC I would not have considered returning to graduate school for the third time, and without her encouragement I would not have believed that I could make it into a success after my previous failures. She has sacrificed a great deal for our little family, giving up her own dreams of graduate school in favor of being a full-time mother while helping me pursue my education and career. Her support, encouragement, and patience have been heroic.

While the research itself may not be earth-shattering or perfect, it is because of the merits and tender mercies of our Savior and Redeemer Jesus Christ that it exists at all. The accomplishment of anything in this life, be it obtaining a degree, overcoming a human weakness, or simply finding the strength to get up and go to work from day to day, is due to His sustaining influence. Without Him I am weak, but in Him I am strong, and He deserves the greatest thanks of all.

Contents

I No Free Lunch for Particle Swarm Optimization 5

III Bayesian Frameworks for Utility-Based Optimization 127

8 The Evolutionary Optimization DBN 129

Chapter 1

Introduction

Optimization is the task of finding a vector in a function's domain that produces the maximum (or minimum) value of its range, a pervasive problem that plays an important role in many disciplines. Frequently the only way to obtain information about the function is through sampling: querying it for values at discrete points in its domain. Many sample-based optimization techniques can be considered in a broad sense to be *evolutionary*: sample values are obtained, information about the function is inferred from those samples, and new sample locations are selected in the hopes of discovering more about the location of the function's global maximum.

Many continuous evolutionary optimization algorithms exist, each of which employs a distinct approach to the selection of sample locations and the way they are used to find the location of the global optimum. Among the more popular approaches are Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995; Shi and Eberhart 1998a; Clerc and Kennedy 2002], Genetic Algorithms (GAs) [Holland 1975; Goldberg 1989; Vose 1999], and Estimation of Distribution Algorithms (EDAs) [Larrañaga et al. 1999; Pelikan et al. 1999; Larrañaga and Lozano 2001], to name a few. Each algorithm has some advantages over the others in particular domains; indeed, this must be the case because of No Free Lunch (NFL) theorems for optimization: no algorithm can have better average performance than random search on all possible functions [Macready and Wolpert 1996; Wolpert and Macready 1997].

While optimization researchers are generally sensitive to the consequences of NFL, i.e., that any algorithm that performs well on one class of functions must perform poorly on the rest, the full impact of the theorems on a given algorithm is generally unknown. In particular, given a traditionally developed optimization algorithm, it is often difficult and usually impossible to obtain a precise definition of the function class on which it will perform well; that it will perform well on one such class is clear, but the exact nature of that class is not. The existence of NFL forces researchers to consider and address this issue, resulting in the common application of awkward empirical approaches (such as the use of benchmark functions) to post-design discovery of the function class [Whitley and Watson 2006].

The application of algorithms to benchmarks is not altogether bad, as it at least establishes common ground for the sake of comparison, but it does little to answer the core question posed by NFL: if an algorithm can only perform well on one class of functions, what is that class?

That this question remains open for traditional approaches to algorithm design represents a serious problem with the way that continuous optimization is perceived in general; when optimization must be done, researchers generally consider the desired *behavior of the algorithm* much more than the *nature of the problem*. Therefore, algorithm design is a process of developing heuristics that achieve certain behavioral characteristics that, while frequently motivated by an implicitly defined function class, leave the pursuit of its precise specification as a subsequent empirical exercise; design and performance desiderata do not match. This causes difficulties for the optimization practitioner who is primarily concerned with selecting an algorithm that is appropriate for a given task: if the algorithm's class is unknown, how can principled algorithm selection be achieved? Answering this question is one of the purposes of this work.

1.1 Thesis Statement

The issues introduced by NFL are not insurmountable, and can be directly addressed by viewing continuous, unconstrained, single-objective optimization for what it is: a problem of selectively gathering and making intelligent use of information to determine the location of the global optimum. In other words, it is a *decision process* where useful samples are selected based on their potential information content or utility, and it is an *inference process* where the information obtained from those samples is combined with assumptions about the function to indicate the location of the global optimum. It therefore makes sense to model the optimization problem and the process by which it is solved using the lingua franca of decision theory: probability density functions, Bayesian inference, and utility.

The central contribution of this work is precisely such a model of the optimization problem, transforming its solution into the process of using well-defined utilities and costs to determine what information should be gathered, then making the most of that information using Bayesian inference. In this model, the function class for which an optimization algorithm is well-suited is not *discovered* as part of a post-design testing procedure, but *declared* by the practitioner as an algorithm prerequisite. This specification requirement elegantly bridges the traditional gap between design and performance desiderata while providing new insights into the nature and impact of NFL for optimization. The resulting algorithm is fixed, intuitive, and powerful, displaying rational and sophisticated behavior when provided with simple and direct specifications.

1.2 Organization

This work is composed of three essential parts, addressing the issues created by No Free Lunch, the role that Bayesian statistics can play in principled evolutionary algorithm design and analysis, and how the problem of optimization may be approached using the tools of decision theory. The first two issues are explored in the context of Particle Swarm Optimization (PSO), and the last introduces a set of novel algorithms.

Part I explores selected domain-specific improvements to PSO, highlighting the fact that each variant implies a different and unknown function class. Part II addresses some deficiencies in the basic PSO mechanism, and it does so by creating a more principled, model-based algorithm design methodology for swarm optimization. Part III generalizes those results, producing the decision-theoretic approach to optimization that forms the core contribution of this work.

Part I

No Free Lunch for Particle Swarm Optimization

Each chapter within Part I is a paper focused on addressing one of the following PSO issues: exposing bias, handling constraints, and premature convergence. Many publications are written to address one or more of them in evolutionary optimization literature.

While not calling direct attention to it, these papers highlight a problem with the way in which evolutionary optimization algorithms are traditionally designed: with no explicit consideration of the function class. Changing an algorithm to improve its performance is really an exercise in altering the class of functions on which it is expected to operate, and such changes are generally serendipitous in nature; having made a change and shown it to be productive for a set of benchmark functions does little to specify the shape of the function class that NFL dictates must be present. That optimization researchers continue to be concerned with making minor alterations to PSO in domain-specific ways is evidence of the fact that the algorithm is implicitly defining a function class that does not include their favorite applications, and the papers in Part I provide examples of the difficulty this presents when attempting to make directed and principled improvements.

In addition to highlighting these principles through the incremental nature of the changes proposed in these papers, the specific problems that each paper addresses are themselves evidence of NFL. Chapter 2, for example, addresses the exposure of origin-seeking bias, showing that the selected test methodology is critical for the exposure of bias; an algorithm commonly believed to be sound is shown to have a significant bias; that this bias went undiscovered for so long is an indication that this important feature of the function class was unknown.

Chapter 3 addresses constraints, but does so in a manner that contrasts with a previously developed PSO extension, an extension that alters PSO such that its effectiveness is drastically reduced; it represents an unwanted alteration of the class of functions for which PSO is well-suited. Had that class been known, it may have been easier to prevent the resulting performance degradation. The approach in this paper restores desirable behavior by allowing unmodified PSO to be applied.

Finally, Chapter 4 addresses an issue that is perhaps the most common in PSO literature: premature convergence. This problem is intimately tied to the notion of the function class assumed by the algorithm: PSO moves particles in such a way as to favor functions that are relatively smooth, and violating that property causes its performance to decrease. Addressing the issue of convergence is a way of enlarging the "smoothness-favoring" function class to include functions that do violate this property to some extent, but a precise definition of that class continues to be unavailable and an approximation must be inferred through experimentation.

In each case, the proposed changes expose the need for knowledge of the function class. In PSO this class specification is buried within its subtle machinery and changes with every minor algorithmic alteration; a more explicit specification is needed and will be provided in subsequent parts of this work.

Chapter 2

Exposing Origin-Seeking Bias in PSO

Published in Proceedings of GECCO 2005, Volume 1, pages 241–248

Abstract

We discuss testing methods for exposing origin-seeking bias in PSO motion algorithms. The strategy of resizing the initialization space, proposed by Gehlhaar and Fogel and made popular in the PSO context by Angeline, is shown to be insufficiently general for revealing an algorithm's tendency to focus its efforts on regions at or near the origin. An alternative testing method is proposed that reveals problems with PSO motion algorithms that are not visible when merely resizing the initialization space.

2.1 Introduction

Particle swarms are now well known as an effective and interesting approach to function optimization. The basic algorithm scatters particles in a limited *feasible region* of the function's domain space, moving them over time in a search for areas of better fitness. Each particle keeps track of a current position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$, as well as the most fit location it has ever seen $\boldsymbol{p}$. The best $\boldsymbol{p}$ among all particles is denoted $\boldsymbol{g}$. In classical PSO, these data are easily combined:

$$\boldsymbol{v}_{t+1} = \chi \left(\boldsymbol{v}_t + \phi_1 \, \mathrm{U}()(\boldsymbol{p} - \boldsymbol{x}_t) + \phi_2 \, \mathrm{U}()(\boldsymbol{g} - \boldsymbol{x}_t) \right) \tag{2.1}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_{t+1} \tag{2.2}$$

where U() is a sample from a standard uniform distribution, ϕ_i are usually somewhere near 2, and χ represents the addition of a constriction coefficient that serves to control the convergence properties of the algorithm [Clerc and Kennedy 2002].

Throughout the remainder of this paper, it will be assumed that minimization is performed. It will also be implied that $\boldsymbol{g}$ refers to the best known position among all $\boldsymbol{p}$ *in a particle's neighborhood*. While this is a slight departure from the norm, where local neighborhood bests have different notation ($\boldsymbol{l}$), the two notations are rarely if ever used in the same context. Therefore, it will be understood that where a notion of sociometry is present, $\boldsymbol{g}$ is the best known among all of the particles in a particular neighborhood and is therefore particle-dependent.

Many population-based optimization approaches, including PSO, suffer from a notable bias: they tend to perform best when the optimum is located at or near the center of the initialization region, which is often the origin. This is especially true when some kind of averaging operator is used to combine information from different members of the population [Angeline 1998]. In many of the standard benchmark functions, the global optimum is at or very near the origin, making this

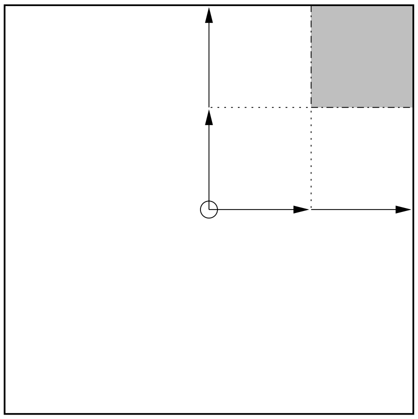


Figure 2.1: Angeline's initialization region for PSO

bias a potential problem when developing and testing a new algorithm. To expose this bias while testing PSO algorithms, Angeline [1998] popularized a method previously introduced by Gehlhaar and Fogel [1996]. The method, hereafter referred to as "Region Scaling" (RS) explicitly excludes the optimum from the initialization region by initializing particles in a new region whose sides are all 1/4 the length of the original, as shown in Figure 2.1.

This paper focuses on test methods used to expose origin-seeking bias in PSO algorithms and shows that RS is not always sufficient. Experiments are done using Clerc's TRIBES [Clerc 2003, 2004] with various kinds of particle motion. TRIBES is described in some detail, followed by descriptions of the kinds of motion chosen for the experiments. An alternative to RS called "Center Offset" (CO) is proposed as a means of exposing the bias in PSO, and experimental results highlight the contrast between the two approaches. Finally, some discussion of the meaning of the results is presented with accompanying recommendations for testing new algorithms.

2.2 TRIBES

TRIBES is a parameter-free approach to swarm size and sociometry in PSO, at the heart of which is a swarm restructuring algorithm which adapts the number of particles and the topology of particle neighborhoods based on swarm performance [Clerc 2003].

Because it is parameter-free, TRIBES provides a useful way to sidestep the issue of swarm size and sociometry specification, providing an out-of-the box approach that has been shown to work well. Because the parameters of swarm size and sociometry are adapted based on the success of the swarm at a given time, TRIBES zeros in rapidly on settings for those parameters that produce the best performance. In the case of a biased motion algorithm, this feature of TRIBES serves to expose that bias effectively.

Clerc's TRIBES paper defines notions of *tribes* and *informers*. A *tribe* is a data structure that keeps track of the particles that belong to it, representing a fully connected subgraph of the overall swarm topology. The *informers* of a particle are itself, all of the particles in its tribe, and any particles of other tribes to which it is connected. All links are symmetrical.

The algorithm begins with one or more particles in a single tribe. The memory of a particle is extended slightly to include not only $\boldsymbol{p}$, but also the number of times it has changed in succession. If the number of successive changes is greater than 0, then improvement was made during the last position update and the particle is labeled "good"[1].

The tribes themselves also receive the labels "good" or "bad", depending on the number of good particles in the tribe. A tribe containing T particles is itself

---

[1] TRIBES also has a notion of "excellent", assigned to a particle if the number is 2 or higher, but we do not make use of that distinction in this paper.

"good" only if U() ≤ G/T, where G is the number of good particles in a tribe and U() is a draw from a standard uniform distribution. Otherwise the tribe is "bad".

Good tribes, because they are doing well and presumably do not need as many particles, will remove one of their particles. Assuming that f is the function being minimized, a good tribe containing more than one particle will remove its worst performer, or the particle with the highest $f(\boldsymbol{p})$. When this occurs, any external links to the particle are reassigned to the best performer in the tribe, i.e. the particle with the lowest $f(\boldsymbol{p})$.

If a good tribe contains only one particle, the tribe itself is removed only if its particle's best external informer has a better $f(\boldsymbol{p})$ than itself. In this latter case, all external links to the particle are reassigned to the external informer.

Bad tribes, on the other hand, presumably need more information, so each creates a new particle *outside of its tribe* and forms a link between the new particle and the best particle within the tribe. The set of all new particles created during one restructuring step forms a *new tribe*. Each new particle is generated randomly and uniformly within the initialization space.

Restructuring occurs once at the beginning of the algorithm and then periodically as it progresses. If, after restructuring, the swarm has N particles and L information links, then restructuring will occur again after $L/2$ swarm iterations, or $NL/2$ function evaluations.

2.3 PSO Motion Algorithms

Several motion algorithms have been suggested for PSO, so many that they cannot all be discussed here. This section describes algorithms that are representative of some interesting features of existing approaches to PSO motion, and these will be used in this paper's empirical study of origin-seeking bias.

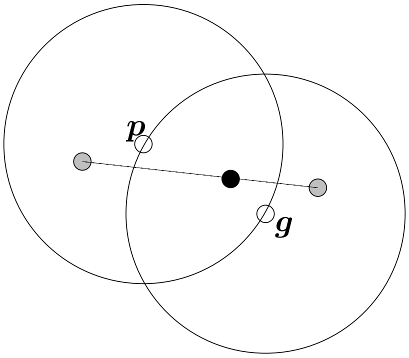


Figure 2.2: TRIBES Pivot method

2.3.1 Pivot

The central motion algorithm introduced in the TRIBES paper is the “Simple Pivot” method. The “Noisy Pivot” method, also introduced in the TRIBES paper, is an extension of Simple Pivot which performs an additional Gaussian sample to generate the final position [Clerc 2003]. We will focus on the Simple Pivot here, referring to it simply as “Pivot”.

The Pivot method, illustrated in Figure 2.2, generates a new position by taking noisy samples in the neighborhood of $\boldsymbol{p}$ and $\boldsymbol{g}$. First, each is taken to be the center of a hypersphere whose radius is $\|\boldsymbol{p} - \boldsymbol{g}\|_2$. Second, a point is sampled from a uniform distribution within each sphere. Each of these samples is given a mass based on the relative fitness of its corresponding center point (either $\boldsymbol{p}$ or $\boldsymbol{g}$), and the new position is the center of mass of the two sampled points.

Mass may be assigned in a number of ways. One simple approach is to assign mass linearly based on the relative fitness of each particle, thus:

$$\boldsymbol{x}_{t+1} = \frac{f(\boldsymbol{p})}{f(\boldsymbol{p}) + f(\boldsymbol{g})} \left(\boldsymbol{g} + \mathrm{U}\left(\|\boldsymbol{p} - \boldsymbol{g}\|_2\right)\right) + \frac{f(\boldsymbol{g})}{f(\boldsymbol{p}) + f(\boldsymbol{g})} \left(\boldsymbol{p} + \mathrm{U}\left(\|\boldsymbol{p} - \boldsymbol{g}\|_2\right)\right) \tag{2.3}$$

where $\mathrm{U}(\cdot)$ is a sample from a hyperspherical uniform distribution with the specified radius. This formula assumes that minimization is occuring, so smaller values of f are favored. Maximization would reverse the positions of the fractional coefficients.

2.3.2 PSOGauss

The second type of motion proposed in conjunction with TRIBES, but not given a name, is based on constricted PSO with Gaussian noise [Clerc 2003]. We will refer to this motion as PSOGauss:

$$\boldsymbol{v}_{t+1} = \chi\left(\boldsymbol{v}_t + \mathrm{G}\left(\boldsymbol{p} - \boldsymbol{x}_t, \frac{1}{4}\mathbf{I}\,\|\boldsymbol{p} - \boldsymbol{x}_t\|_2^2\right) + \mathrm{G}\left(\boldsymbol{g} - \boldsymbol{x}_t, \frac{1}{4}\mathbf{I}\,\|\boldsymbol{g} - \boldsymbol{x}_t\|_2^2\right)\right) \tag{2.4}$$

where $\mathbf{I}$ is the identity matrix, $\chi \approx 0.71441$, and $\mathrm{G}(\cdot,\cdot)$ is a sample from a Gaussian distribution parameterized by the supplied mean and covariance matrix.

This approach is similar to constricted PSO and different from the others in this section because it uses velocity instead of computing a position directly.

2.3.3 BareBones

The BareBones motion algorithm is probably the simplest PSO algorithm proposed to date, but it is very successful at optimization [Kennedy 2003]. The motion equation is given here:

$$\boldsymbol{x}_{t+1} = \mathrm{G}\left(\frac{1}{2}(\boldsymbol{p} + \boldsymbol{g}), \mathbf{I}\,\|\boldsymbol{p} - \boldsymbol{g}\|_2^2\right) \tag{2.5}$$

where $\mathrm{G}(\cdot,\cdot)$ is a sample from a Gaussian distribution parameterized on a mean and covariance matrix.

This approach was developed after noting that the distribution of samples between $\boldsymbol{p}$ and $\boldsymbol{g}$ was distinctly Gaussian at each time step in classical PSO and was an effort to cull out any useless properties of the traditional motion equations.

This particular method was not the only one proposed in Kennedy's Bare Bones paper [2003], but it is the simplest and is very effective.

2.4 Experiments

To test for origin seeking behavior, the following benchmark functions were used:

Sphere:

$$f(\boldsymbol{x}) = \sum_{i=1}^{D} x_i^2 \qquad R = (-50, 50)^D \tag{2.6}$$

Rastrigin:

$$f(\boldsymbol{x}) = \sum_{i=1}^{D} x_i^2 + 10 - 10\cos(2\pi x_i) \qquad R = (-5.12, 5.12)^D \tag{2.7}$$

Rosenbrock:

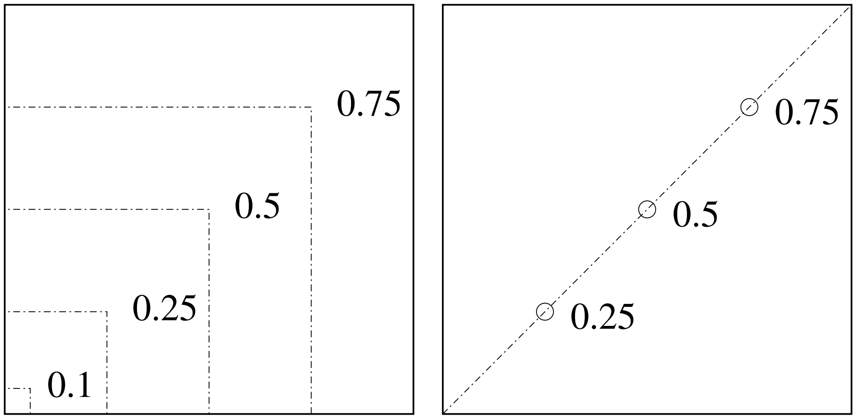
$$f(\boldsymbol{x}) = \sum_{i=1}^{D-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \qquad R = (-100, 100)^D \tag{2.8}$$

Sphere is unimodal and symmetric, Rastrigin is highly multimodal and symmetric, and Rosenbrock is multimodal and asymmetric. These functions are representative of the essential characteristics of a number of popular benchmarks.

For each function and type of motion, experiments were performed using "Region Scaling" (RS) and "Center Offset" (CO). In the first (RS), several different initialization regions were chosen, each formed by taking a fraction of the feasible rectangle in each dimension as shown in Figure 2.3(a)[2]. In the second (CO), the center of each function was moved to a different location of space, as shown in Figure2.3(b), leaving the initialization region in its original location. For example,

---

[2] This is somewhat different from Angeline's approach, since the region is chosen from the opposite corner. While this does not affect the symmetric Sphere or Rastrigin functions, it tends to initialize particles in a more challenging part of Rosenbrock's domain.



(a) Region Scaling (RS) (b) Center Offset (CO)

Figure 2.3: Different methods of exposing origin-seeking bias

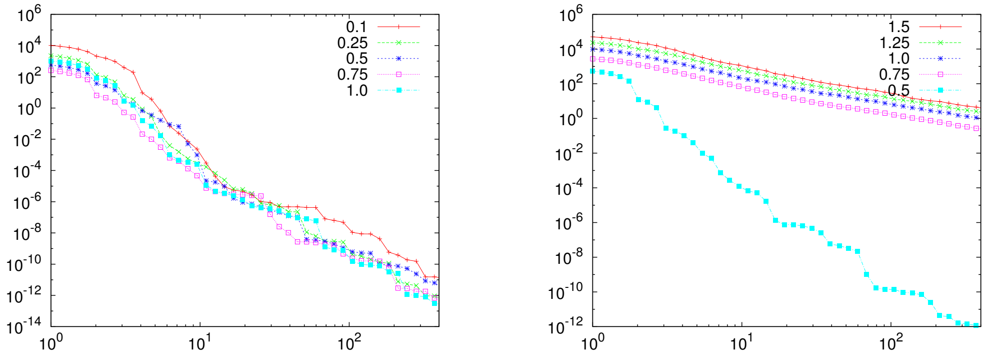
Sphere would become

$$f(\boldsymbol{x}, \boldsymbol{c}) = \sum_{i=1}^{D} (x_i - c_i)^2$$

where $\boldsymbol{c}$ is the location of the new center, calculated using the feasible region and the numbers shown in the figure (0.5 leaves the center unchanged). This can only be applied to functions whose support extends outside of the feasible region, as is the case with all of the benchmarks used here. CO values outside of the range $[0, 1]$ are valid and indicate that the center has moved beyond the boundaries of the feasible region along the line shown.

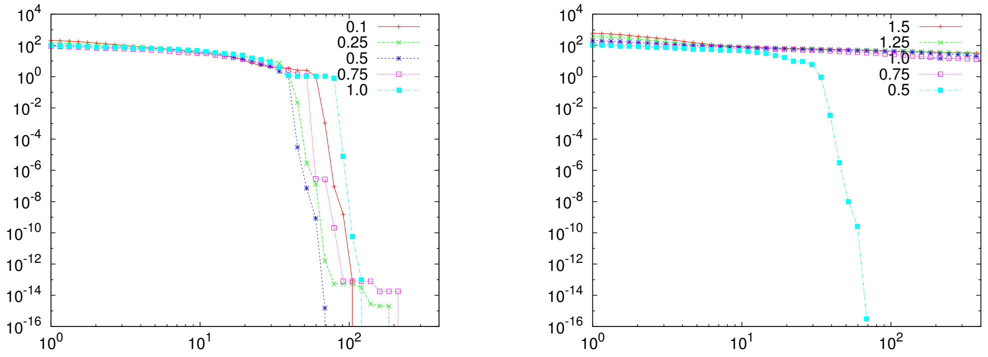
2.5 Results

The results of all of the experiments are shown in Figures 2.4, 2.5, and 2.6. The x-axis of each graph shows one tic per 50 function evaluations, and the y-axis is the best fitness obtained among all particles. Because the focus is minimization, lower values are better. All results are averaged over 30 runs and plotted on a log-log scale.



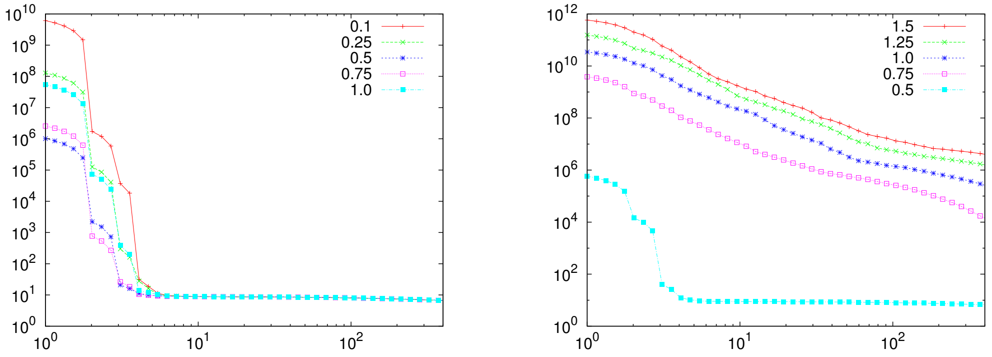
(a) Sphere iterations (RS)

(b) Sphere iterations (CO)



(c) Rastrigin iterations (RS)

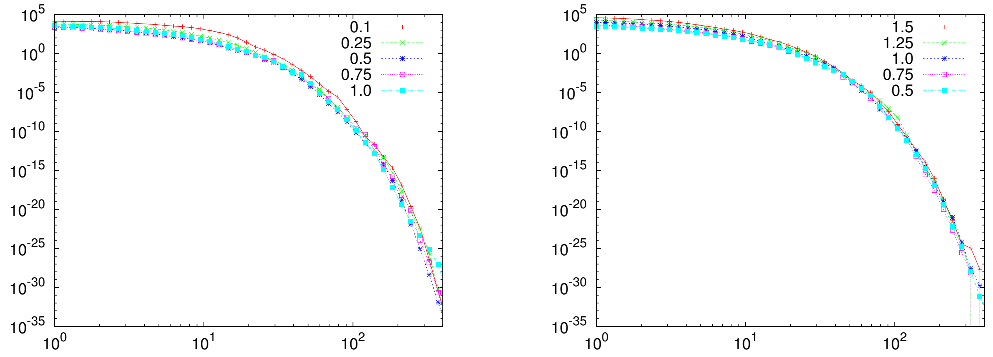
(d) Rastrigin iterations (CO)



(e) Rosenbrock iterations (RS)

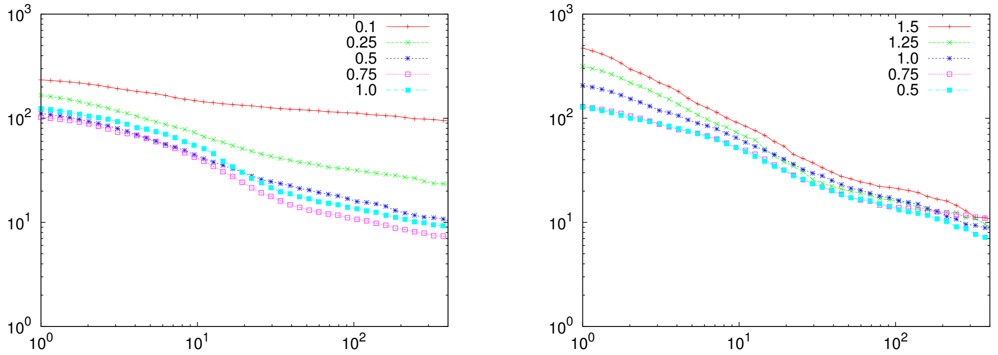
(f) Rosenbrock iterations (CO)

Figure 2.4: Pivot performance under Region Scaling and Center Offset



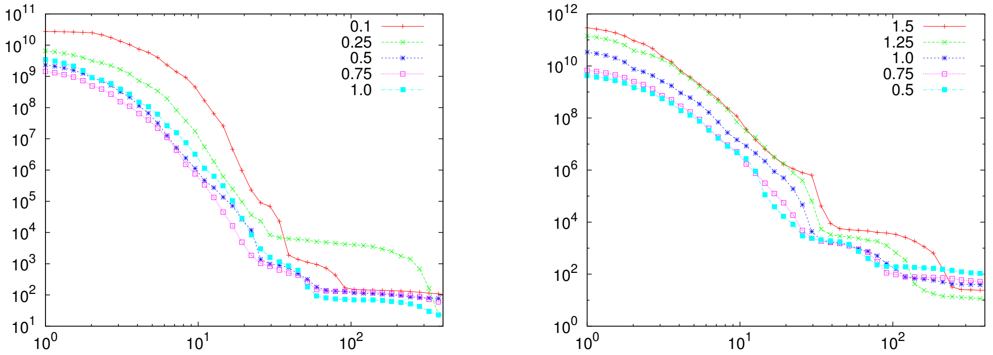
(a) Sphere iterations (RS)

(b) Sphere iterations (CO)



(c) Rastrigin iterations (RS)

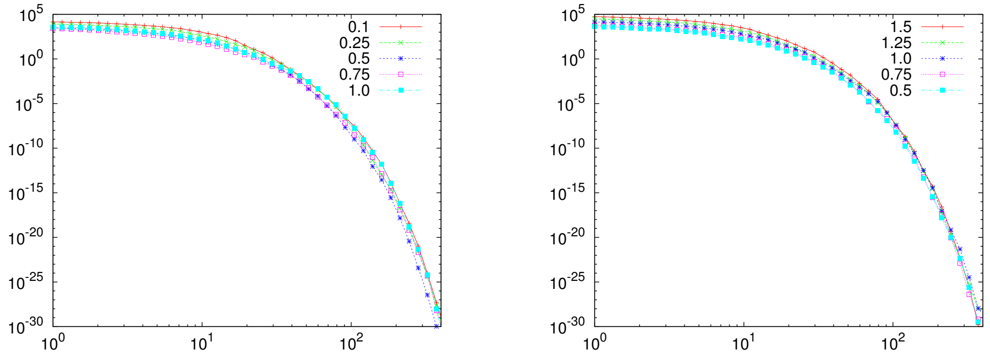
(d) Rastrigin iterations (CO)



(e) Rosenbrock iterations (RS)

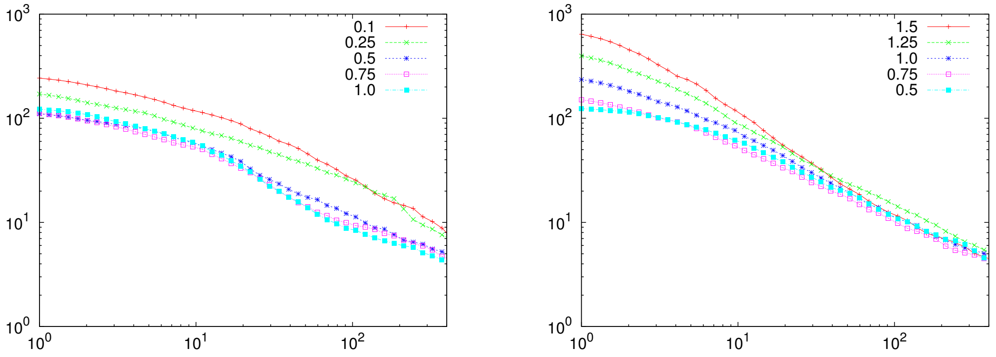
(f) Rosenbrock iterations (CO)

Figure 2.5: PSOGauss performance under Region Scaling and Center Offset



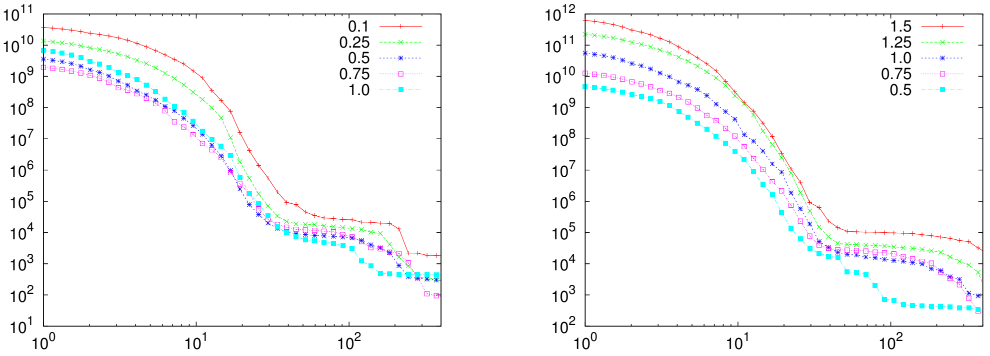
(a) Sphere iterations (RS)

(b) Sphere iterations (CO)



(c) Rastrigin iterations (RS)

(d) Rastrigin iterations (CO)



(e) Rosenbrock iterations (RS)

(f) Rosenbrock iterations (CO)

Figure 2.6: BareBones performance under Region Scaling and Center Offset

Figure 2.4 shows the results of the experiments using the Pivot method. Using RS the Pivot method appears to perform equally well in all cases, easily overcoming the difficulties imposed by a smaller initialization region. When CO is applied, however, the bias becomes evident. Pivot only performs well when the global minimum is located at or near the origin.

Figure 2.5 displays results for PSOGauss. In this case, the origin seeking bias is more subtle. The key is to look for natural grouping of results near the end of a run. With Rastrigin and RS, some clustering occurs among the regions that include the global minimum in Figure 2.5(c), but it does not show up when using CO in Figure 2.5(d). Rosenbrock and Sphere show no significant clustering in either case.

In Figure 2.6 the results are shown for BareBones. Similar to PSOGauss, clustering is observed with Rastrigin and RS in Figure 2.6(c) but not with CO in Figure 2.6(d). On Rosenbrock, however, clustering is definitely observed (note the log-log scales) both under RS in Figure 2.6(e) and CO in Figure 2.6(f). The clustering observed is much more striking under CO, with some counterintuitive results under RS. Again, no such clustering is observed when tested on Sphere.

2.6 Discussion

2.6.1 Exposure Methods

The behavior of Pivot in Figure 2.4 suggests that a strong argument can be made for using CO to test for origin-seeking bias; it succeeded where RS failed. Additionally, when looking at results for Rosenbrock among all motion algorithms, anywhere that RS exposed a bias, CO did as well. In that sense, CO appears to be no worse than RS, and in the case of Pivot it is vastly better for discovering bias.

The Rastrigin case is somewhat different, where any bias shown on that function only occurred under RS. While this may say more about Rastrigin than any of the algorithms used to optimize it, it does expose weakness in both PSOGauss and BareBones when dealing with such a highly regular and multimodal function.

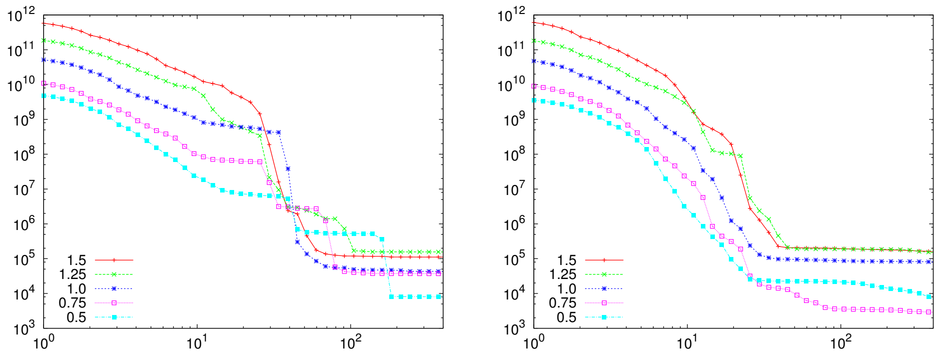
The types of bias exposed by these two approaches are different. RS exposes a bias toward the *center of the initialization region*, while CO exposes a bias *toward the absolute origin*. To further verify this idea, other experiments were performed that moved the center of the function and the initialization region by the same amount simultaneously. This simply performed a coordinate shift for the entire problem, something that would not be expected to cause difficulty for any of the algorithms here. The results for Pivot, however, were nearly identical to those shown here when using CO exclusively, indicating that there is indeed a bias toward the absolute origin in that algorithm.

It is possible to combine both the RS technique and the CO technique into a single experiment, shifting the coordinate system and then shrinking the initialization region. This approach can sometimes expose both kinds of bias at once, suggesting that if only one experiment is to be done, RS and CO should be combined. Otherwise, it is best to do each separately in order to expose the various potential algorithmic weaknesses.

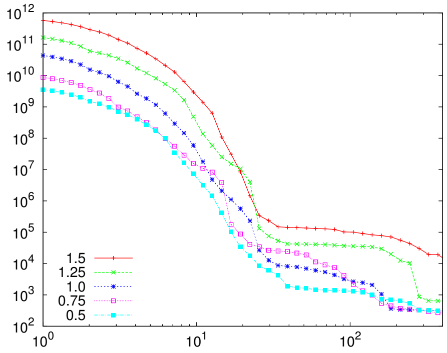
2.6.2 TRIBES Behavior

The use of TRIBES as the basis for swarm size and sociometry, while not an arbitrary choice, merits further discussion. It was mentioned previously that TRIBES was chosen because it tends to find the right combination of sociometry and swarm size for effectively exposing the bias in a motion algorithm. It does this because it adapts swarm characteristics based on performance.

The biased behavior does not only show up when using TRIBES, however. Figure 2.7 gives results for BareBones on various fixed-size fully connected swarms. The experiments are performed under CO. It is especially clear in Figures 2.7(a)



(a) Rosenbrock iterations – 6 Particles (b) Rosenbrock iterations – 8 Particles



(c) Rosenbrock iterations – 10 Particles

Figure 2.7: BareBones average fitness using CO, the star sociometry, and various fixed swarm sizes

and 2.7(b) that BareBones displays origin-seeking bias on Rosenbrock. It is therefore possible to find the behavior using a fixed swarm size and a specific sociometry, but it can be difficult to find the right combination by hand. More particles implies initially more diverse function samples and increases the likelihood of finding a good area to explore at the beginning of the run, making it difficult to see any bias that may exist. Fewer particles have little available information and therefore nearly *always* get stuck quickly, making it difficult to make any convincing statements about observed bias. TRIBES seems to get it just right.

That TRIBES is good at exposing the bias actually makes a very positive statement about the algorithm in general. The exposure occurs because it is using just enough particles and just enough connections between them to get the best results possible. In other words, when an algorithm is center seeking, it *exploits* that fact because it finds a good combination of swarm size and sociometry for that algorithm. It makes sense to test new algorithms using TRIBES because strange origin-seeking behavior may otherwise be masked by lucky choices of fixed swarm size and sociometry.

2.6.3 Benchmark Behavior

Rastrigin is somewhat unique among the benchmarks here in that it exposes a bias only under RS. The others expose it either in both cases or only in the CO case. Why does this happen with Rastrigin? The function is highly multimodal with very deep local minima spread out on a *regular grid*. In order for particles to find the global minimum, they must jump over or out of these local minima in a reliable way. If particles manage to acquire the correct speed, they tend to jump quickly from one minimum to the next since a straight line will pass through many evenly-spaced local minima. A larger initialization space facilitates the discovery of an appropriate velocity while a smaller one tends to generate particles that get stuck because of small initial velocities.

Rosenbrock has some counterintuitive behavior. When applying RS, the bias appears to be reversed in some cases. This may be due to its asymmetric properties; too many particles in a misleading area of the space (with strange local minima) can cause the swarm to converge too quickly to a challenging part of the domain. As the initialization region is made smaller but still includes the global minimum, fewer particles start out in misleading areas.

These properties do not discredit these benchmarks as indicators of bias, but rather highlight some of the unique issues that they expose. It is a good idea to use multiple different benchmarks when looking for bias.

2.6.4 Motion Algorithms

Pivot is undeniably biased, but what of the others? Between PSOGauss and BareBones, PSOGauss appears to display the least bias, since it works well on Rosenbrock no matter what is done to it. BareBones, on the other hand, appears to show significant bias on Rosenbrock. Both show a small amount of bias on Rastrigin when the region size is altered.

If they must be ranked, then, it appears that PSOGuass is the least biased, followed by the slightly more biased BareBones, finally followed by the extremely biased Pivot.

What it is about Pivot and BareBones that makes them biased is not obvious from the results. They are different from PSOGauss in one very fundamental way, however: they update positions directly while PSOGauss updates velocities. This one difference may be enough to account for a center seeking bias, though that idea has not yet been fully explored. The reasons behind the extreme nature of Pivot's bias also merit further exploration.

2.7 Conclusion

Region Scaling (RS), a popular method of testing for bias in PSO, is effective but not always sufficient for detecting origin-seeking behavior. In fact, on motion like the Pivot method, it fails to expose any bias whatsoever. Center Offset (CO), on the other hand, catches cases that are not otherwise visible, making it an essential testing tool for any new PSO algorithm.

Additionally, it was found that TRIBES provides a useful framework for testing different kinds of PSO motion, given that it tends to exploit the best behavior of a motion algorithm. This, in combination with CO and RS is a very effective method of testing for origin-seeking bias.

Chapter 3

Linear Equality Constraints and Homomorphous Mappings in PSO

Published in Proceedings of CEC 2005, Volume 1, pages 73–80

Abstract

We present a homomorphous mapping that converts problems with linear equality constraints into fully unconstrained and lower-dimensional problems for optimization with PSO. This approach, in contrast with feasibility preservation methods, allows any unconstrained optimization algorithm to be applied to a problem with linear equality constraints, making available tools that are known to be effective and simplifying the process of choosing an optimizer for these kinds of constrained problems. The application of some PSO algorithms to a problem that has undergone the mapping presented here is shown to be more effective and more consistent than other approaches to handling linear equality constraints in PSO.

3.1 Introduction

Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995] is a social algorithm that is most naturally applied to unconstrained optimization problems. Potential solutions called 'particles' are initialized within and 'flown' through the target function's domain, searching for the global minimum or maximum. The standard formulas for particle motion are given as follows:

$$\mathbf{v}_{t+1} = \omega \mathbf{v}_t + \phi_1 \mathrm{U}_{\mathbf{1}t} \otimes (\mathbf{p} - \mathbf{x}_t) + \phi_2 \mathrm{U}_{\mathbf{2}t} \otimes (\mathbf{g} - \mathbf{x}_t) \tag{3.1}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1} \tag{3.2}$$

where ω is the *inertia weight*, each $\phi_i \approx 2$, each $\mathrm{U}_{\mathbf{i}}$ is a vector of numbers drawn from a standard uniform distribution, and $\otimes$ performs point-wise vector multiplication [Mendes et al. 2004]. The variables $\mathbf{p}$ and $\mathbf{g}$ are different for each particle and represent the best known position in the particle's own past and the best known position among particles in its neighborhood, respectively.

While unconstrained optimization is the process of finding a vector $\mathbf{g}^\star \in \mathbb{R}^D$ such that $f(\mathbf{g}^\star)$ is the global optimum, constrained optimization involves finding an optimal $\mathbf{g}^\star \in \mathcal{F}$ where $\mathcal{F} \subset \mathbb{R}^D$ is a feasible subspace of the original domain. In other words, the addition of constraints always restricts the space from which an optimal vector may be taken. A number of approaches have been used to add constraint-handling capabilities to PSO, each different depending on the nature of the constraints [Paquet and Engelbrecht 2003a; Coath and Halgamuge 2003; Hu and Eberhart 2002; Parsopoulos and Vrahatis 2002; Pulido and Coello 2004].

Though diverse in detail, the various methods of handling constraints in evolutionary optimization algorithms can be categorized as one or more of the following [Koziel and Michalewicz 1999; Paquet and Engelbrecht 2003a]:

Preserve: All potential solutions are initialized within $\mathcal{F}$ and special operators are applied to search for new solutions without violating the constraints.

Penalize: The fitness of solutions not in $\mathcal{F}$ is artificially reduced in some way to make those solutions less desirable.

Partition: Solutions are partitioned into feasible and infeasible sets and each set is treated differently. This includes techniques such as repair of infeasible solutions and prioritizing solutions based on feasibility.

Preprocess: The problem itself is transformed so that the constraints are either easier to handle or eliminated. The Homomorphous Mapping introduced by Koziel and Michalewicz [1999] is in this category.

This paper is concerned with improving the performance of PSO when applied to problems with linear equality constraints. These constraints are generally given in the form $\mathbf{Ax} = \mathbf{b}$. Admittedly, linear equality constraints form a very small subset of possible constraints, but they appear in useful real world problems such as the training of support vector machines [Paquet and Engelbrecht 2003b]. Some interesting work specific to handling linear equality constraints in PSO is found in the Linear PSO (LPSO) and Converging Linear PSO (CLPSO) algorithms introduced by Paquet and Engelbrecht [2003a].

These algorithms, while simple to implement and empirically effective, have two basic limitations. First, they rely on feasibility preservation, which inherently restricts algorithm design because the constraints must define the set of possible motion operators. Second, the linear restriction placed on the motion equations is known to reduce the effectiveness of PSO in unconstrained problems [Monson and Seppi 2004], leading one to ask whether another style of PSO motion may be more effective in linearly constrained problems.

We propose a homomorphous mapping that transforms a space constrained by $\mathbf{Ax} = \mathbf{b}$ into a space that is not only fully unconstrained, but also of lower dimensionality, allowing any unconstrained optimization algorithm to be directly applied to a much easier problem. We begin by discussing LPSO and CLPSO, followed by the introduction of the homomorphous mapping suitable for handling linear equality constraints and some discussion about the motivation for the algorithm. Results comparing existing constrained optimization techniques are then given.

3.2 LPSO and CLPSO

LPSO (Linear PSO) is much like classical PSO, except that rather than use a different random number for each element of the velocity and position vectors, a single scalar is multiplied by each vector, thus:

$$\mathbf{v}_{t+1} = \omega\mathbf{v}_t + \phi_1 \mathrm{U}_{1t}(\mathbf{p}_i - \mathbf{x}_t) + \phi_2 \mathrm{U}_{2t}(\mathbf{g}_i - \mathbf{x}_t) \quad . \tag{3.3}$$

This means that the resultant velocity (and therefore position) is a strictly linear combination of other particle positions. If the particles are all initialized within $\mathcal{F} = \{\mathbf{x} | \mathbf{Ax} = \mathbf{b}\}$, then they will always be within $\mathcal{F}$. CLPSO is similar to LPSO except that its globally best particle has its own motion equation: its next position is calculated as the sum of its personal best and a small random velocity within the null space of the equality constraints. This allows the swarm to do more local exploration and guarantees that at least a local minimum will be found.

These algorithms fill an interesting gap in the constrained PSO literature because they focus solely on linear equality constraints. They also have the advantage of relative implementation simplicity. While CLPSO appears to have much better exploration capabilities than LPSO, however, both are based upon a version of PSO that has observably poor exploration characteristics as the particles near convergence.

Consider for a moment the problem of unconstrained optimization using LPSO. If we think of the positions of particles as vectors, some or all of which participate in a basis set, then linear combinations of these vectors will span a space. Motion that is a result of strictly linear operations of these positions will force particles to always be within that span; this fact is what makes LPSO a *feasibility preserving* method.

This same feature, however, cripples it in terms of exploration capability. If there are fewer particles than effective constrained dimensions, then the algorithm is doomed from the start to explore a space with lower effective dimensionality than the target domain. If there are more particles than effective dimensions, they must be initialized in such a way as to span the entire target domain, something that is fairly likely when using random initialization. Even when this is the case, however, as some particles approach convergence and diversity decreases in the swarm, fewer of the positions will be sufficiently unique to contribute to a full span, and the dimensionality of the searchable space decreases quickly.

In either case, the search space is eventually overconstrained. This behavior of reduced search dimensionality can be observed when watching LPSO near convergence. As some particles become still, the rest will increasingly explore along a periodic straight line trajectory through $\mathbf{g}$. This exploration strategy can work well on some functions like Rastrigin, where the local minima are spread out on a regular grid, but in general it is not effective.

Even the use of some diversity increasing approaches like ARPSO [Riget and Vesterstrøm 2002] or Spatial Extension PSO [Krink et al. 2002] does little to solve the problem, as these are commonly implemented to perform a linear change to the particle's motion. As long as the underlying motion overconstrains the search space, these diversity increasing methods are of little help.

The overconstraining of the problem over time results in premature convergence to locations of the target domain that are not even local minima, an issue that motivated the development of CLPSO (Converging Linear PSO), which changes the motion equation for the best particle in the swarm so that it explores in a complete span of the feasible domain using a random velocity component in the null space of $\mathbf{A}$. This idea is mathematically sound and empirically effective, but it is possible that fundamentally changing the underlying motion will produce better results.

3.3 Homomorphous Mappings in PSO

The homomorphous mapping approach proposed by Koziel and Michalewicz [1999] has many advantages over preservation methods like LPSO and CLPSO, not least of which are the ability to use an unmodified unconstrained optimization algorithm and a sometimes significant reduction of the dimensionality of the problem. This idea is especially interesting when using PSO, since it is simple to implement, effective at optimization, and most naturally applied to unconstrained problems.

In general terms, the goal of a homomorphous mapping is to convert a difficult constrained problem into a simpler constrained or unconstrained problem. The burden of constraint handling is thus shifted from the optimization algorithm to an algorithm that creates a *transform* or *decoder* $\mathcal{H} : \mathcal{S} \mapsto \mathcal{F}$ such that $\mathcal{S}$ is a space that is easier to work with than $\mathcal{F}$. The use of the decoder allows an optimization algorithm to work with points $\mathbf{x} \in \mathcal{S}$ while evaluating the target function in its original space: $f(\mathcal{H}(\mathbf{x}))$. For more information on this interesting idea, see Koziel and Michalewicz [1999].

3.3.1 Linear Equality Constraints

Linear equality constraints of the form $\mathbf{Ax} = \mathbf{b}$ always define a hyperplane, assuming that the rows of $\mathbf{A}$ are linearly independent [Paquet and Engelbrecht 2003a]. Since a hyperplane has lower effective dimensionality (D^-) than the space in which it exists (D), it is always possible to reorient the plane such that it is completely contained within $\mathbb{R}^{D^-}$, a space that is spanned by a subset of the axes in $\mathbb{R}^D$. For example, a *plane* in $\mathbb{R}^3$ can always be oriented to lie in the x–y plane, and a *line* in $\mathbb{R}^3$ can be oriented to lie along the x axis.

The size of $\mathbb{R}^{D^-}$ may be easily determined from the linear equality constraints themselves. Each row of $\mathbf{A}$ represents a vector that is normal to a hyperplane in $\mathbb{R}^D$, and the effective dimensionality of this hyperplane is always $D - 1$. To illustrate this idea, it is useful to think of adding constraint hyperplanes into a space one at a time; the first hyperplane reduces the effective dimensionality by 1, the second forms an intersection with the first which drops another effective dimension, and so on. The goal of the homomorphous mapping is to reorient the resulting lower-dimensional hyperplane such that it is contained entirely within $\mathbb{R}^{D^-}$, allowing search to be restricted to that smaller unconstrained space during optimization.

The most obvious such mapping is a *projection* from the larger space into the lower dimensional space, but this has some disadvantages, such as the necessity of selecting out the appropriate dimensions in order to perform a useful (non-degenerate) projection. The mapping on which we will focus our attention in this paper is composed of rotations and translations which are represented in a single homogeneous matrix $\mathbf{H} = \mathbf{T}^{-1}$. The complete method for calculating $\mathbf{T}$ is given in Algorithm 1 and a more detailed explanation follows.

Algorithm 1 HHM(*pairs*)

```
$\mathbf{T} = \mathbf{I}$
# Rotate Space
for $i = 1$ to len($pairs$) do
    $a = D - (i - 1)$
    $\mathbf{p}_1, \mathbf{p}_2 = pairs[i]$
    for $j = 1$ to $a - 1$ do
        $\mathbf{n}^+ = \mathbf{T}(\mathbf{p}_2 - \mathbf{p}_1)^+$
        $\theta = \text{atan2}(n_j, n_a)$
        $\mathbf{T} = \mathbf{R}_{\theta,j,a}\mathbf{T}$
    end for
end for
# Translate Space
for $i = 1$ to len($pairs$) do
    $a = D - (i - 1)$
    $\mathbf{p}_1, \mathbf{p}_2 = pairs[i]$
    $\tilde{\mathbf{p}}_1^+, \tilde{\mathbf{p}}_2^+ = \mathbf{T}\mathbf{p}_1^+, \mathbf{T}\mathbf{p}_2^+$
    $\mathbf{n} = \tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1$
    $\mathbf{q} = (\tilde{\mathbf{p}}_1 \cdot \mathbf{n})\mathbf{n}$
    if $q_a \neq 0$ then
        $T_{a,D} = -\left\|\mathbf{q}\right\|^2 / q_a$
    end if
end for
return $\mathbf{T}$
```

3.3.2 Homogeneous Homomorphous Mapping (HHM)

Because each row of $\mathbf{A}$ and each corresponding element of $\mathbf{b}$ together form the equation of a hyperplane, the constraint system $\mathbf{A}\mathbf{x} = \mathbf{b}$ may be rewritten as a set of equations of the form $\mathbf{n}_i \cdot \mathbf{x} = b_i$, where $\mathbf{n}_i$ is normal to plane i and b_i is a distance parameter. If $\mathbf{n}_i$ is of unit length, then b_i has a convenient geometric interpretation: it is the distance from the origin to the plane in the direction of $\mathbf{n}_i$ as illustrated in Figure 3.1. The figure also shows a useful alternative definition of a plane using two points:

$$\mathbf{p}_1 = b\mathbf{n} \tag{3.4}$$

$$\mathbf{p}_2 = \mathbf{p}_1 + \mathbf{n} \ . \tag{3.5}$$

This two-point definition of a hyperplane is used in Algorithm 1, which requires that each $\mathbf{n}$ has unit length. Since any constraint system $\mathbf{Ax} = \mathbf{b}$ may be trivially rewritten to satisfy this requirement, it will be assumed throughout the rest of this paper that constraints are normalized in this way.

The HHM algorithm is composed of two high-level steps. Starting at line 2 it calculates all of the necessary rotations that will orient the constraint hyperplane so that it is *parallel* to $\mathbb{R}^{D^-}$, but not necessarily contained within it. On line 9 it begins the process of finding the translation that will move the hyperplane so that it has no support outside of $\mathbb{R}^{D^-}$.

Each of these steps will be given special consideration below. The result of the algorithm is a homogeneous matrix of the form

$$\mathbf{T} = \begin{pmatrix} r_{1,1} & \cdots & r_{1,D} & t_1 \\ \vdots & \ddots & \vdots & \vdots \\ r_{D,1} & \cdots & r_{D,D} & t_D \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{3.6}$$

where $r_{i,j}$ participates in rotation and t_i participates in translation. A vector multiplied through this matrix must also be homogeneous (augmented with a terminal 1):

$$\mathbf{p}_1^+ = \begin{pmatrix} p_{1,0} & \cdots & p_{1,D} & 1 \end{pmatrix}^\top . \tag{3.7}$$

The value *pairs* required by the HHM algorithm is a list of point pairs representing the constraint planes as defined in (3.4) and (3.5).

To better describe the HHM algorithm, which applies to arbitrary linear equality constraints in any number of dimensions, it is useful to work through a concrete example where the number of constraints and the dimensionality are fixed. The discussion that follows will assume that $\mathbf{A}$ has two rows and that $D = 3$.

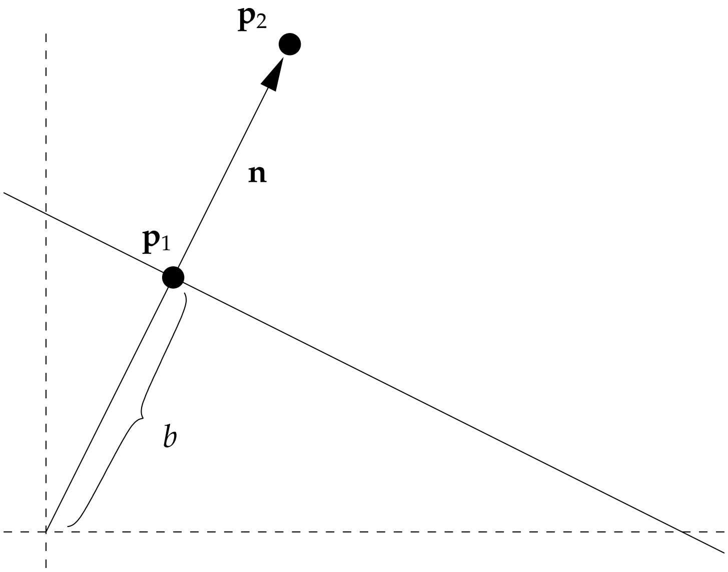


Figure 3.1: A plane defined by unit normal $\mathbf{n}$ and distance b, or by the points $\mathbf{p}_1$ and $\mathbf{p}_2$

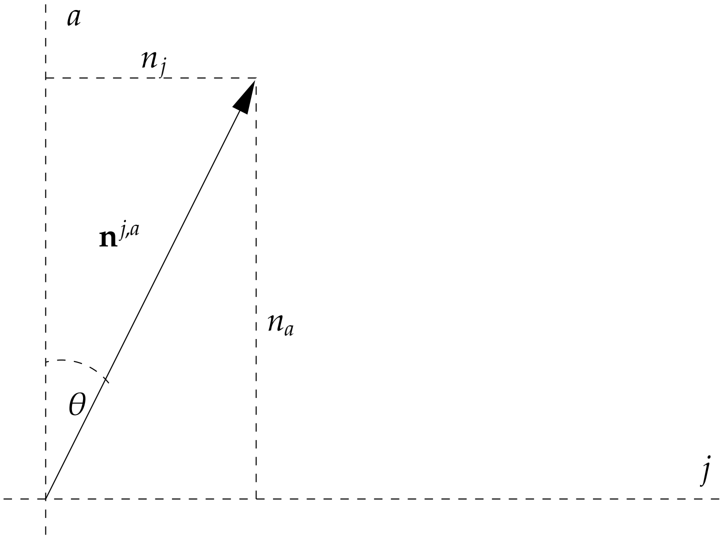


Figure 3.2: Calculating the rotation for a normal projection

Each constraint represents a plane in $\mathbb{R}^3$, and the two constraints together form a line at their intersection. The HHM algorithm will be applied to create a transform **T** that orients the entire space so that this line lies along the x-axis.

Rotation

The first step is to rotate each plane so that the intersection is parallel to the x axis. We begin with plane 1, which is defined by two points $\mathbf{p}_1, \mathbf{p}_2 = pairs[1]$. This plane will be rotated so that its normal is parallel to the z axis, effectively eliminating the need to consider that axis during subsequent rotations.

The first plane is realigned by rotating a projection of the normal in two planes, starting with the x–z plane; the normal is transformed so that its projection in the x–z plane (denoted $\mathbf{n}_1^{x,z}$) lies along the z axis. Once this is done, the projection will be oriented correctly, but the actual normal vector may still have some support along the y axis.

Figure 3.2 illustrates what the algorithm is doing on lines 6–7: it first gets the normal into the current space and then calculates the rotation angle θ that will cause the projection of the normal into the j–a plane to lie along the a axis. In this example, j is the index of the x component in **n** and a is the index of the z component in **n**.

The angle θ is computed by

$$\theta = \arctan \frac{n_j}{n_a} \tag{3.8}$$

and is used to construct a rotation matrix $\mathbf{R}_{\theta,j,a}$ that will orient $\mathbf{n}_1^{x,z}$ along the z axis. The rotation matrix is the identity matrix of size $D + 1$ with the exception of the following elements:

$$\begin{aligned} R_{j,j} &= \cos\theta & \qquad R_{j,a} &= -\sin\theta \\ R_{a,j} &= \sin\theta & \qquad R_{a,a} &= \cos\theta \ . \end{aligned}$$

Again referring to the example, the process is repeated in the y–z plane so that $\mathbf{n}_1^{y,z}$ lies along the z axis. When finished, the unprojected $\mathbf{n}_1 = (0 \ \ 0 \ \ 1)^\top$ and is therefore lined up along the z axis.

Because the first plane now has constant support along the z axis, the intersection of the planes does as well. Therefore, when applying this process to subsequent planes, that axis need not be considered again. When the next plane is considered, a rotation is performed in the x–y plane so that the projection of the second normal into that plane ($\mathbf{n}_2^{x,y}$) lines up with the y axis. When that is done, the intersection of the two planes will be parallel with the x axis, and the rotation step is complete.

Translation

Because the planes may not have crossed through the origin, the rotation step does not limit the constraint hyperplane to $\mathbb{R}^{D^-}$. It does, however, have *constant* support in all but the first D^- dimensions. The last step performed by the HHM algorithm performs a translation so that this constant support is removed, e.g. the intersection of the two planes in $\mathbb{R}^3$ is not merely parallel to the x-axis, but superimposed over it.

This translation step is made more convenient by the two-point definition of a plane shown in Figure 3.1. Translating so that the first plane contains the origin is very simple: its normal points along the z-axis and therefore it needs to be translated by its (easily calculated) distance from the origin. Once this step

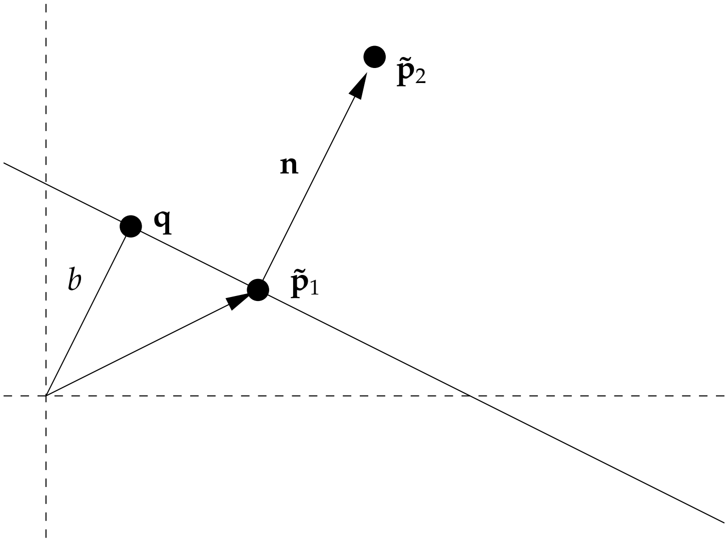


Figure 3.3: After translation, **q** is calculated

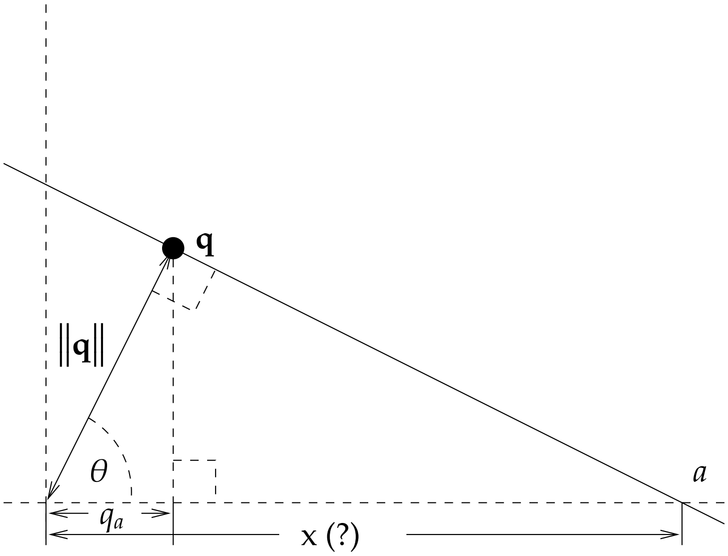


Figure 3.4: Calculating the final translation for a hyperplane

is complete, however, the second plane may look something like that shown in Figure 3.3. Note that $\tilde{\mathbf{p}}_1$ is no longer the point in the plane closest to the origin, and therefore $\left\|\tilde{\mathbf{p}}_1\right\| \neq b$.

Fortunately, it is easy to calculate the point in the plane closest to the origin using a dot product: $\tilde{\mathbf{p}}_1 \cdot (\tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1) = b$. The point $\mathbf{q}$ closest to the origin is simply $b(\tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1)$ as calculated in lines 13–14 of Algorithm 1. Given $\mathbf{q}$ it is possible to calculate the amount of y axis translation necessary to ensure that plane 2 contains the origin. As long as no translation occurs in the z axis, the first plane will still contain the origin as well.

Figure 3.4 illustrates the way in which the translations are calculated. The angle θ is part of two triangles, and can therefore be used in two formulas to find the unknown distance x:

$$\cos\theta = q_a / \left\|\mathbf{q}\right\| \tag{3.9}$$

$$\cos\theta = \left\|\mathbf{q}\right\| / x \tag{3.10}$$

producing

$$x = \left\|\mathbf{q}\right\|^2 / q_a \tag{3.11}$$

which is how the translation is calculated in lines 15–16. This calculation works for every plane to which it is applied, including the first.

3.3.3 Comments on HHM

The result of applying HHM is a homogeneous matrix $\mathbf{T}$ that transforms *every point* in $\mathbb{R}^D$ to *another point* in $\mathbb{R}^D$. Importantly, points in the feasible region $\mathcal{F}$ are all transformed by this process to be contained within $\mathbb{R}^{D^-}$, e.g. no vector in

$\mathcal{F}$ will have a nonzero value for y or z after applying **T**, effectively reducing the dimensionality of the target function.

To obtain the desired matrix $\mathcal{H} : \mathbb{R}^{D^-} \mapsto \mathbb{R}^{D}$, one need merely invert **T** and appropriately pad vectors in $\mathbb{R}^{D^-}$ with zeros and a trailing 1 before multiplying. Performing a general inverse operation, however, is unnecessary because of the nature of rotation matrices; it is straightforward to obtain the inverse by splitting out the rotation and translation components of **T**. The effects of applying **H** can then be obtained by first applying the negative of the translation vector followed by the transposed rotation matrix. Other optimizations are possible, but are beyond the scope of this paper.

It is natural to ask why something simple like Gaussian Elimination was not used instead of this rotation/translation mapping. The advantages of the HHM presented here are that it preserves Euclidean distance and it produces an easily-reversed mapping, fulfilling two of the desiderata for homomorphous mappings [Koziel and Michalewicz 1999]. Gaussian Elimination, on the other hand, performs a *projection* and is difficult to implement in a numerically stable way in all cases; in order to apply Gaussian Elimination in a way that is guaranteed to be stable, one must choose the appropriate subset of axes on which to do the projection (equivalent to determining the way in which columns of **A** are reordered), hopefully in such a way that distances in the projection correspond to similar distances in the original space [Koziel and Michalewicz 1999]. The HHM does this automatically by preserving Euclidean distance, and its potential numerical problems inherent in repeated matrix multiplication are easily addressed by infrequent reorthonormalization.

3.4 Experiments

Given the above algorithm for calculating a mapping, handling linear constraints is as simple a task as finding **H** and searching using particles $\mathbf{x} \in \mathbb{R}^{D^-}$ while evaluating $f(\mathbf{Hx})$ in the original space. The approach outlined here is actually more general than its application to PSO, since any unconstrained optimization procedure may be applied after $\mathcal{H}$ has been calculated.

3.4.1 Experimental Setup

Several benchmark functions were applied with the introduction of LPSO and CLPSO, comparing them against Genocop II, an evolutionary optimization package [Michalewicz 1996]. These benchmarks are also commonly used to test unconstrained optimization algorithms:

$$
\begin{aligned}
\text{Sphere}(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 \\
\text{Quadratic}(\mathbf{x}) &= \sum_{i=1}^{D} \sum_{j=1}^{D} e^{-(x_i - x_j)^2} x_i x_j + \sum_{i=1}^{D} x_i \\
\text{Rastrigin}(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 + 10 - 10\cos(2\pi x_i) \\
\text{Rosenbrock}(\mathbf{x}) &= \sum_{i=1}^{D-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \\
\text{Griewank}(\mathbf{x}) &= \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad .
\end{aligned}
$$

Here they are subject to the following linear equality constraints [Paquet and Engelbrecht 2003a]:

$$\mathbf{A} = \begin{pmatrix} 0 & -3 & -1 & 0 & 0 & 2 & -6 & 0 & -4 & -2 \\ -1 & -3 & -1 & 0 & 0 & 0 & -5 & -1 & -7 & -2 \\ 0 & 0 & 1 & 0 & 0 & 1 & 3 & 0 & -2 & 2 \\ 2 & 6 & 2 & 2 & 0 & 0 & 4 & 6 & 16 & 4 \\ -1 & -6 & -1 & -2 & -2 & 3 & -6 & -5 & -13 & -4 \end{pmatrix} \tag{3.12}$$

$$\mathbf{b} = \begin{pmatrix} 3 & 0 & 9 & -16 & 30 \end{pmatrix}^{\top} \quad . \tag{3.13}$$

Using the mapping produced by HHM, results were obtained by applying the following *unconstrained* implementations of PSO to the resulting lower-dimensional problems:

Constricted:

$$\mathbf{v}_{t+1} = \chi\left(\mathbf{v}_t + \phi_1 U_{\mathbf{1}t} \otimes (\mathbf{p} - \mathbf{x}_t) + \phi_2 U_{\mathbf{2}t} \otimes (\mathbf{g} - \mathbf{x}_t)\right) \tag{3.14}$$

BareBones:

$$\mathbf{x}_{t+1} = G\left(\frac{1}{2}(\mathbf{p} + \mathbf{g}), \mathbf{I} \left\|\mathbf{p} - \mathbf{g}\right\|_2^2\right) \tag{3.15}$$

PSOGauss:

$$\mathbf{v}_{t+1} = \chi\left(\mathbf{v}_t + G\left(\mathbf{p} - \mathbf{x}_t, \frac{1}{4}\mathbf{I} \left\|\mathbf{p} - \mathbf{x}_t\right\|_2^2\right) + G\left(\mathbf{g} - \mathbf{x}_t, \frac{1}{4}\mathbf{I} \left\|\mathbf{g} - \mathbf{x}_t\right\|_2^2\right)\right) \tag{3.16}$$

Constricted PSO [Clerc and Kennedy 2002] used $\phi_1 = \phi_2 = 2.05$ with $\phi = \phi_1 + \phi_2$ and $\chi = 2/|2 - \phi - \sqrt{\phi^2 - 4\phi}|$. BareBones [Kennedy 2003] is a simple parameter-free algorithm proposed by Kennedy, and PSOGauss is a version of Constricted PSO with Gaussian noise as proposed by Clerc in his TRIBES paper [2003]. In

Table 3.1: Sphere performance after 250 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	304.884	387.746	37.612	1680	54.846	16.939	32.544	107.584
LPSO	445.316	803.006	**32.137**	4505	**32.137**	7×10^{-12}	**32.137**	**32.137**
CLPSO	32.139	0.007	**32.137**	32.183	**32.137**	3×10^{-6}	**32.137**	**32.137**
Constricted	**32.137**	2×10^{-10}	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**
BareBones	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**
PSOGauss	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**

Table 3.2: Quadratic performance after 1000 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	49.945	10.996	35.393	82.221	39.5	9.785	35.41	56.613
LPSO	758.525	1496	35.4	11230	59.762	39.831	**35.377**	246.905
CLPSO	68.57	53.865	**35.377**	196.067	39.832	10.887	**35.377**	71.38
Constricted	**36.165**	**3.117**	**35.377**	**55.538**	35.783	2.394	**35.377**	55.538
BareBones	40.019	9.609	**35.377**	75.147	37.079	5.332	**35.377**	55.538
PSOGauss	38.998	8.59	**35.377**	72.482	**35.589**	**0.528**	**35.377**	**36.892**

the definitions of both BareBones and PSOGauss, $G(\cdot, \cdot)$ produces a draw from a multivariate Gaussian distribution with the supplied mean and covariance. In all unconstrained algorithms, a star sociometry is used.

3.4.2 Results

Tables 3.1–3.5 duplicate Paquet and Engelbrecht's results using Genocop II, LPSO, and CLPSO [Paquet and Engelbrecht 2003a]. The tables also provide the results of applying HHM to the three unconstrained algorithms above. Except on the Rastrigin function, the use of the HHM allows *all* of the unconstrained algorithms to outperform not only LPSO and CLPSO, but Genocop II as well. Genocop II has better worst-case performance on Rastrigin but only has better average performance when employing 20 particles.

On every benchmark, including Rastrigin, the unconstrained algorithms find minima that are at least as good as those found by the constrained algorithms. Notably, every unconstrained algorithm has better best and worst-case behavior than the constrained algorithms on Griewank.

Table 3.3: Rastrigin performance after 1000 generations

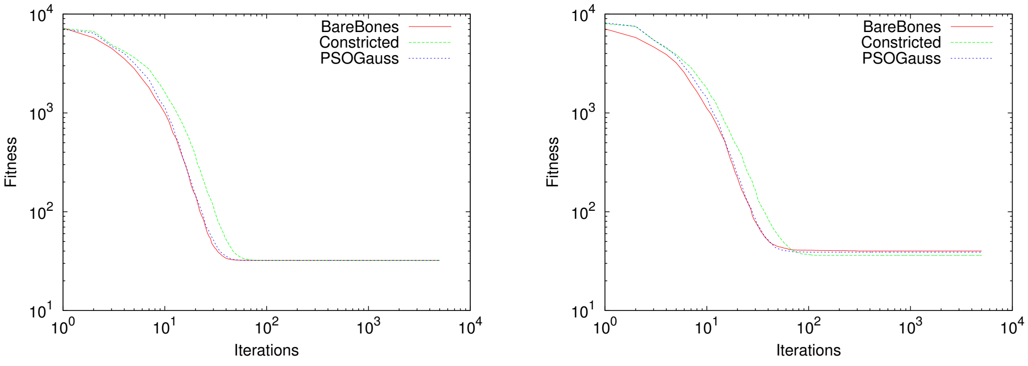
	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	52.379	**7.498**	37.116	**67.564**	**43.059**	**6.142**	37.011	**59.959**
LPSO	76.487	30.699	**36.975**	232.979	75.011	27.719	38.965	184.226
CLPSO	69.039	21.591	**36.975**	154.379	76.896	27.304	**36.975**	151.394
Constricted	**50.431**	12.314	**36.975**	85.728	46.199	7.477	**36.975**	76.736
BareBones	55.921	16.06	**36.975**	119.556	49.238	10.191	**36.975**	76.774
PSOGauss	55.622	14.826	**36.975**	119.094	47.11	8.136	**36.975**	68.802

Table 3.4: Rosenbrock performance after 2000 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	21630	154.443	21490.8	22031	21485.7	0.4	21485.4	21486.6
LPSO	4×10^6	2×10^7	21554.2	2×10^8	126000	1×10^6	21485.9	1×10^7
CLPSO	744600	7×10^6	**21485.3**	7×10^7	**21485.3**	9×10^{-8}	**21485.3**	**21485.3**
Constricted	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**
BareBones	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**
PSOGauss	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**

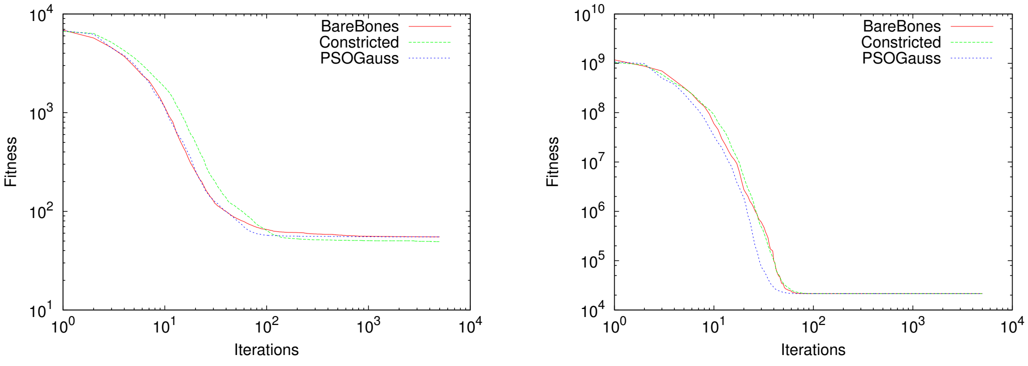
Table 3.5: Griewank performance after 1000 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	0.702	0.187	0.417	0.971	0.584	**0.131**	0.201	0.843
LPSO	2.997	2.945	0.387	15.805	1.695	1.921	0.338	14.401
CLPSO	3.049	2.101	0.236	16.427	1.9	2.379	0.236	17.259
Constricted	**0.488**	**0.168**	**0.151**	**0.83**	**0.413**	0.145	**0.151**	**0.792**
BareBones	0.523	0.181	0.203	0.912	0.444	0.158	**0.151**	0.83
PSOGauss	0.53	**0.168**	**0.151**	0.958	0.454	0.174	**0.151**	0.83



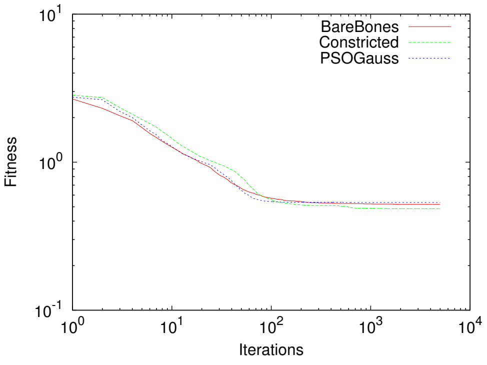
(a) Sphere

(b) Quadratic



(c) Rastrigin

(d) Rosenbrock



(e) Griewank

Figure 3.5: Average fitness over time for unconstrained optimizers with HHM and 10 particles

In addition to these results, Figure 3.5 shows the average fitness obtained by the swarm over time. The average is computed over 100 runs using 10 particles. These graphs show that *every* unconstrained algorithm (using HHM) on every benchmark has converged to good values by the time 100 generations have completed. Time did not allow for the creation of similar experiments with LPSO, CLPSO, and Genocop II (this should be done in the future), but it is useful to know that good values may be obtained earlier from the HHM method than the tabulated data suggest.

3.5 Conclusions

The homomorphous mapping is a useful and effective alternative to feasibility preservation when dealing with linear equality constraints in PSO. The particular mapping developed here, the HHM, is simple to implement, does not suffer from the numeric problems inherent in using Gaussian Elimination, and allows the application of any unconstrained optimization algorithm to a problem of reduced dimensionality. The performance of the unconstrained PSO algorithms chosen here is not only better than that of both LPSO and CLPSO in many instances, it also compares favorably with or outperforms Genocop II.

The ability to apply any unconstrained optimization algorithm to functions with linear equality constraints is a benefit by itself, since there are many more effective unconstrained optimization algorithms than those that handle constraints directly, many of which have been well tuned. Reducing the problem dimensionality provides further benefits that cannot be ignored.

The HHM approach described here may also be useful when working with linear *inequality* constraints; it is possible that it could form the basis for a truly general method of linear constraint handling. Work is ongoing in this area and will be addressed more completely in the future.

It remains to be seen how this approach will fare in real world applications like the training of SVMs, a potentially interesting direction for future research.

Chapter 4

Adaptive Diversity in PSO

To Appear in Proceedings of GECCO 2006

Abstract

Spatial Extension PSO (SEPSO) and Attractive-Repulsive PSO (ARPSO) are methods for artificial injection of diversity into particle swarm optimizers that are intended to encourage converged swarms to engage in exploration. While simple to implement, effective when tuned correctly, and benefiting from intuitive appeal, SEPSO behavior can be improved by adapting its radius and bounce parameters in response to collisions. In fact, adaptation can allow SEPSO to compete with and outperform ARPSO. The adaptation strategies presented here are simple to implement, easy to tune, and retain SEPSO's intuitive appeal.

4.1 Introduction

Particle Swarm Optimization (PSO) is a social or evolutionary optimization algorithm that was discovered during experiments with simulated bird flocking [Kennedy and Eberhart 1995]. Its discovery has led to an algorithm which has gained popularity in recent years for its simplicity, relatively small number of tuning parameters, and surprising effectiveness on a large class of functions.

Classical PSO begins by scattering particles in the function domain space, often by means of a uniform distribution bounded by a function-specific region of feasibility. Each particle is a data structure that maintains its current position $\mathbf{x}$ and its current velocity $\dot{\mathbf{x}}$. Additionally, each particle remembers the most fit position it has obtained in the past, denoted $\mathbf{p}$ for "personal best". The most fit $\mathbf{p}$ among all particles is written $\mathbf{g}$ for "global best".

A valuable variant on classical approaches is *constricted PSO*, where each particle updates its state using the following equations (written in a slightly non-traditional way to accentuate the role of acceleration):

$$\ddot{\mathbf{x}}_{t+1} = \phi_1 \, \mathrm{U}() \otimes (\mathbf{p} - \mathbf{x}_t) + \phi_2 \, \mathrm{U}() \otimes (\mathbf{g} - \mathbf{x}_t) \tag{4.1}$$

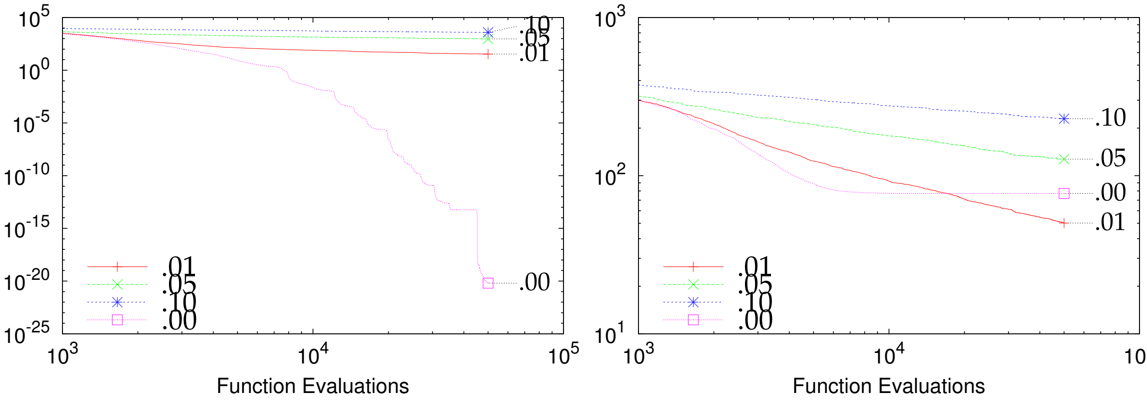
$$\dot{\mathbf{x}}_{t+1} = \chi \, (\dot{\mathbf{x}}_t + \ddot{\mathbf{x}}_{t+1}) \tag{4.2}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \dot{\mathbf{x}}_{t+1} \tag{4.3}$$

where $\phi_1 = \phi_2 = 2.05$, U() is a vector whose elements are drawn from a standard uniform distribution, and $\otimes$ represents element-wise multiplication. The constriction coefficient χ is in this case defined to be

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{4.4}$$

where $\kappa = 1.0$ and $\phi = \phi_1 + \phi_2$ [Clerc and Kennedy 2002].



(a) Sphere (b) Rastrigin

Figure 4.1: Spatial Extension PSO (SEPSO) with multiple radius settings

Though effective, PSO sometimes suffers from premature convergence on problems with many local minima. Convergence is in general a desirable property, allowing the swarm to search regions near the global minimum at increasing levels of detail as time progresses. Unfortunately, in the context of many local minima, the convergence property may cause a swarm to become trapped in one of them and fail to explore more promising neighboring minima.

Designers of optimization algorithms therefore face a fundamental tradeoff: search the current local minimum in detail through quick convergence, or consume resources exploring other areas of the domain [Riget and Vesterstrøm 2002]. In an effort to handle this tradeoff more explicitly in PSO, some notable diversity-increasing approaches have been proposed. One such approach, the Spatial Extension PSO (SEPSO), involves endowing each particle with a radius, then causing particles to bounce off of one another [Krink et al. 2002]. A related approach, called Attractive-Repulsive PSO (ARPSO), measures the global diversity of the swarm, triggering modes of global attraction or repulsion when it crosses predefined thresholds [Riget and Vesterstrøm 2002]. Though effective when well-tuned, finding good function-specific tuning parameters for these methods is non-trivial.

The tuning parameters in SEPSO and ARPSO alike represent a threshold that dictates when diversity will be artificially added to either a single particle or to the swarm as a whole, respectively. Especially in the case of SEPSO, the threshold is easier to tune and the algorithm's performance improves when a simple adaptation strategy is applied.

We begin by describing SEPSO and demonstrating the issues implicit in setting its radius parameter. We then describe the proposed adaptation methodology used to improve robustness of parameters and performance on multimodal functions. We then briefly describe ARPSO, a successor to SEPSO that is less amenable to improvements using our adaptation strategy and that rarely outperforms the easily implemented SEPSO extensions presented here.

4.2 Spatial Extension PSO

The Spatial Extension PSO (SEPSO) is a simple method of artificially injecting diversity into a swarm. While in classical PSO, particles are conceptually volumeless and therefore never collide with one another, the basic premise of SEPSO is that particles have a spherical volume that is defined by a radius r. Two particles i and j collide when

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq 2r \ . \tag{4.5}$$

In the event of a collision, the involved particles "bounce" backwards, effectively moving to a point that is formed by reflecting the intended current position about the previous position, optionally reversing the velocity as well to create a post-bounce position $\mathbf{x}'_{t+1}$ and velocity $\dot{\mathbf{x}}'_{t+1}$:

$$\dot{\mathbf{x}}'_{t+1} = -\dot{\mathbf{x}}_{t+1} \tag{4.6}$$

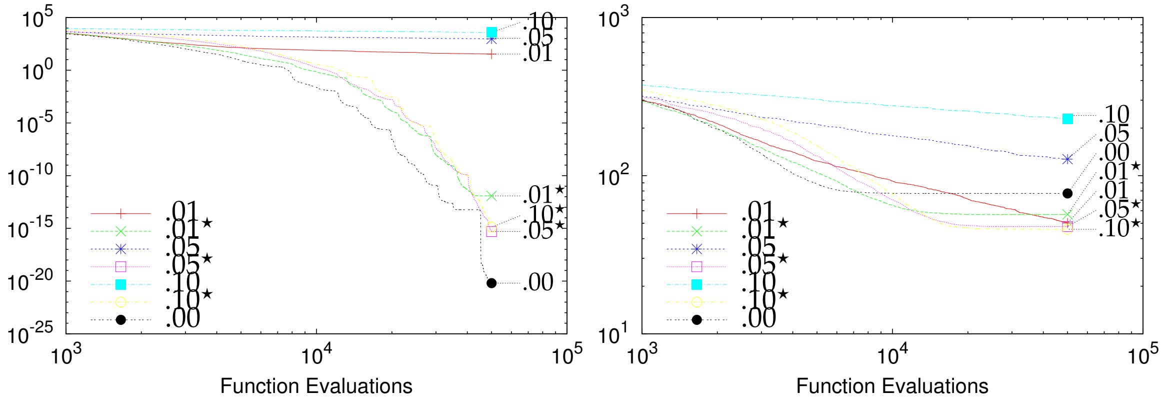
$$\mathbf{x}'_{t+1} = \mathbf{x}_t - (\mathbf{x}_{t+1} - \mathbf{x}_t) \ . \tag{4.7}$$

This approach is simple to implement and has intuitive appeal: if a particle is very close to its neighbors, it is likely to be duplicating work by exploring regions that are covered by other particles and should therefore move away from them. The combination of a radius with associated notions of collisions and bouncing is an effective and intuitive way to accomplish this goal.

This method of increasing diversity is also appealing because it can be applied to nearly any variant of PSO, including those that do not have an explicit notion of velocity (e.g. Bare Bones PSO [Kennedy 2003]): the new location is calculated according to the specified PSO algorithm, then tested against all other new locations; if a collision occurs, that location is reflected before the particle's state changes. Again, velocity may optionally be reversed when present.

The choice of radius r, though not addressed in the original SEPSO work [Krink et al. 2002], is critical to the performance of the algorithm. Consider Figure 4.1, which illustrates the relative performance of different radius settings. The radius is set to a constant fraction of the length L of the longest diagonal of the feasible regions for Sphere and Rastrigin (defined in Table 4.1). Unless otherwise stated, all figures are generated by averaging 30 runs with constricted PSO as the baseline motion, $D = 30$ dimensions, a fully-connected swarm of size 20, and velocity reversal in the event of SEPSO collisions.

The figure matches intuition. On the simple unimodal function Sphere, for which PSO is already an efficient optimizer, bouncing can only slow down desirable convergence, thereby hurting performance. As the collision radius is



(a) Sphere

(b) Rastrigin

Figure 4.2: Contracting Radius SEPSO (CRS) with fixed and adaptive$^{(\star)}$ radius settings

decreased, performance gets closer to that achieved by the baseline motion. On the highly multimodal Rastrigin, however, bouncing can be helpful, avoiding the stagnation to which PSO is generally prone for such functions. In this case, a setting of $r = .01L$ represents an improvement over baseline PSO, and the trend that is evident in the radius setting seems to indicate that a smaller radius would provide even better performance. This trend cannot continue indefinitely, however, as setting the radius to 0 simply reproduces the behavior of classical PSO. Finding a good setting for the radius is therefore a problem-dependent exercise; multiple runs may be required to obtain a useful value.

4.2.1 Adaptive Radius

Convergence, as previously discussed, is a desirable property for PSO, since particles will tend to explore small regions in greater detail as they begin to move more slowly and to converge on a single point in space. This detailed exploration can be important since the scale of the global minimum may not be known before

PSO is applied. SEPSO, unfortunately, frequently prevents not only premature convergence but useful and appropriate convergence as well.

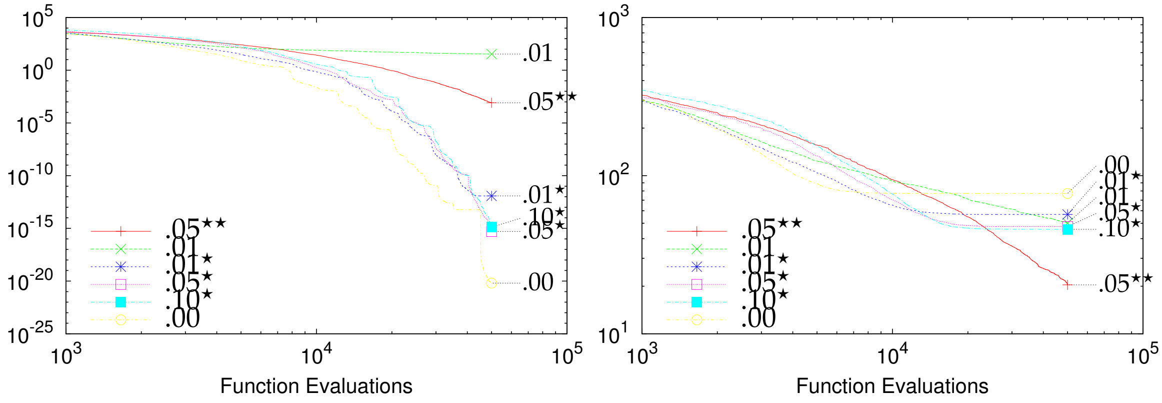
This is entirely due to the fact that the radius is fixed: whatever else the particles may be doing, they always bounce when within a predefined distance ($2r$) of one another, effectively limiting the scale of the space that may be searched: if they are trying to explore a detailed region of space but are thwarted by collisions, that region will remain unexplored unless a fortunate accident occurs.

Both problems are addressed by giving each particle an individual, adaptable radius. In this case, the detection of a collision causes particles to bounce as before, but the radius of colliding particles is also decreased to make bouncing less likely in the future. This allows particles to escape local minima into which they may become trapped while admitting exploration at increasing levels of detail as time progresses. This idea can be implemented by defining a global adaptation constant $\gamma \in [0,1]$ and an individual bounce count b for each particle. Each particle's bounce count is initialized to 0 and is incremented whenever the particle is involved in a collision. Collision between particles i and j occurs when

$$\|\mathbf{x}_i - \mathbf{x}_j\| \leq (\gamma^{b_i} + \gamma^{b_j})r \quad . \tag{4.8}$$

No change is made to (4.6) or (4.7), leaving bouncing mechanics intact and introducing negligible computational overhead. Adapting the radius in this way results in a new algorithm: the "Contracting Radius SEPSO" (CRS). Results of this approach with $\gamma = 0.8$ and various radius settings are demonstrated in Figure 4.2, and more will be given later. The superscript $\star$ indicates an adaptive result.

Note that when applied to Sphere, CRS performs more closely to the baseline ($r = 0$), which is not unexpected. The radius decreases every time a particle bounces, making it less likely to collide with other particles as time progresses. As



(a) Sphere

(b) Rastrigin

Figure 4.3: Contracting Radius, Increasing Bounce SEPSO (CRIBS) with adaptive radius$^{(\star)}$ and distance$^{(\star\star)}$

a result, the swarm regains the ability to converge, though it does so more slowly than before. Notice also that on Rastrigin the adaptive versions all perform better than the baseline and are clustered more closely (note especially the log scale) around lower values than their non-adaptive counterparts.

Unfortunately, the adaptive version still suffers from premature convergence on Rastrigin. Standard SEPSO with $r = 0.01L$ not only eventually overtakes all of the adaptive versions, it also continues on a downward trend.

4.2.2 Adaptive Distance

CRS's observed premature convergence behavior is present in other multimodal functions, prompting an additional extension to CRS: the "Contracting Radius, Increasing Bounce SEPSO" (CRIBS), where individual bounce distance is also adapted. Although various bounce distances have been attempted by the SEPSO authors without noticeable improvement [Krink et al. 2002], individually increasing particle bounce distance while decreasing collision radius has merit; as particles converge, their diversity decreases and the locations to which they bounce will tend to be in the same local minimum. Therefore, as the adaptive radius *decreases*, a good indicator for convergence, the bounce distance should *increase* to make the act of bouncing more effective. This is accomplished through a simple change to (4.7):

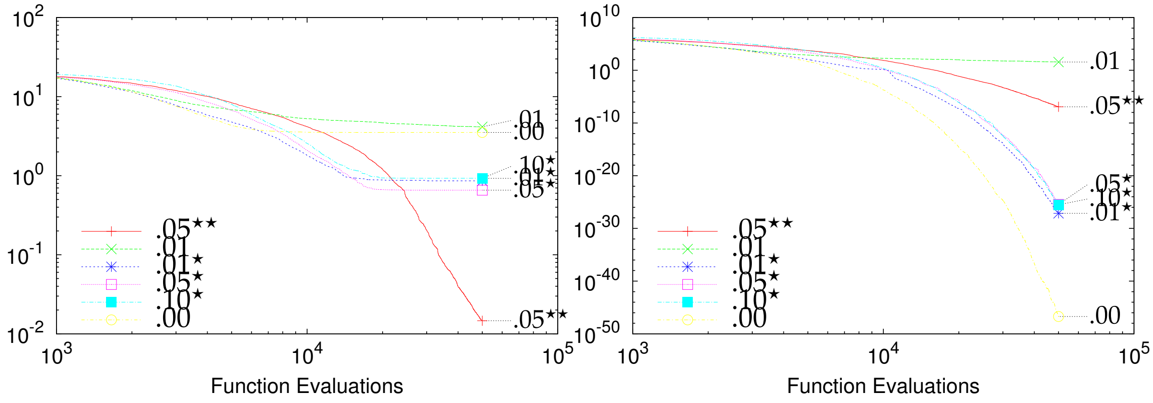
$$\mathbf{x}'_{t+1} = \mathbf{x}_t - \gamma^{-b}(\mathbf{x}_{t+1} - \mathbf{x}_t) \tag{4.9}$$

In other words, while the radius is *decreased* via multiplication by γ^b, the distance is similarly *increased* by γ^{-b}. Employing a bounce distance that is inversely proportional to collision radius may be expected to hurt performance on unimodal functions by wasting function evaluations on distant points; however, it should be expected to improve performance on multimodal functions by increasing each particle's odds of escaping a local minimum. These predictions are verified in Figure 4.3 where it is shown that performance suffers for Sphere while significantly improving for Rastrigin.

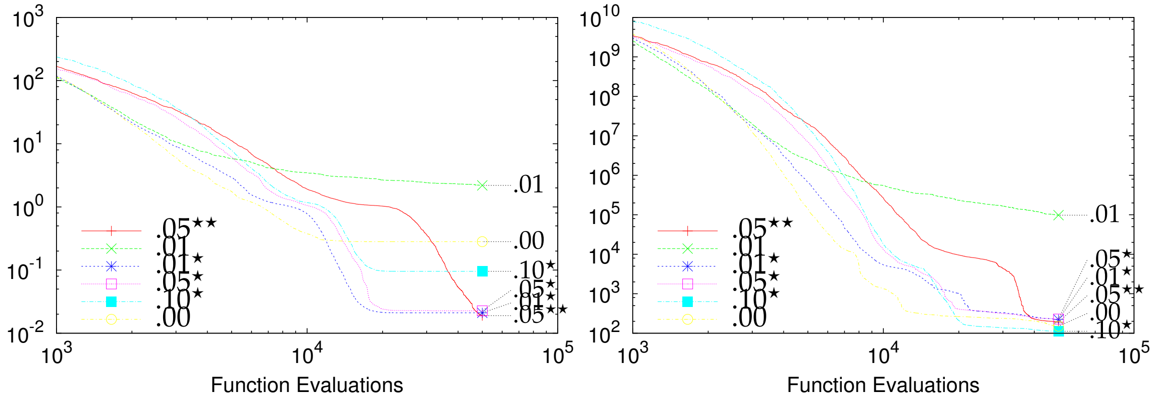
It should be noted that while only one radius setting is shown for CRIBS to avoid clutter, far more data were collected than can be presented in this setting. Those data make it clear that the initial radius becomes less important when adaptation is present; on Rastrigin, for example, CRIBS always outperformed CRS by a large margin.

4.2.3 Remarks and Additional Results

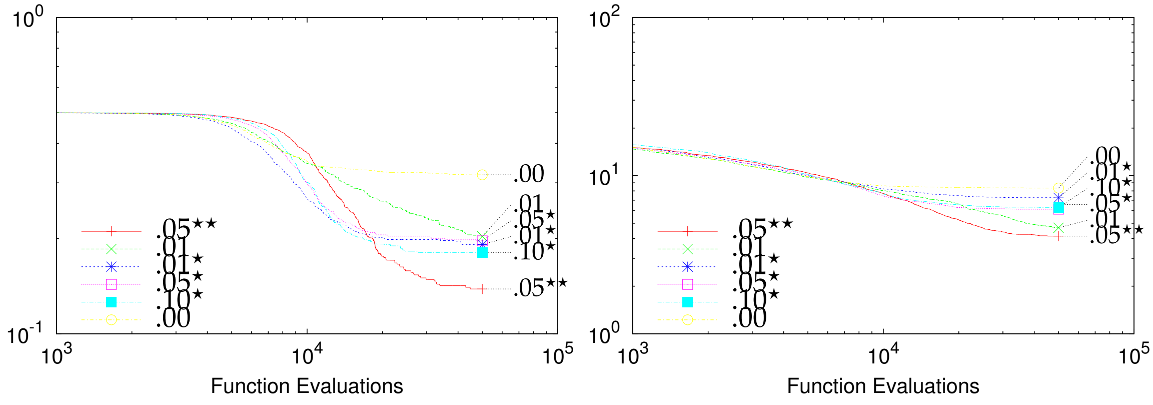
The robustness of the initial parameter settings is affected by adapting those parameters over time. In the case of SEPSO, the radius setting has a dramatic impact on the performance of the algorithm, and it is clear in Figure 4.1 that the parameters selected for the experiment are not low enough for Sphere but are beginning to approach appropriate values for Rastrigin. Adaptation, however, makes the choice of initial radius far less important, as illustrated in Figure 4.2. In each case, adapting



(a) Ackley (b) DeJongF4



(c) Griewank (d) Rosenbrock



(e) SchafferF6 (f) SchafferF7

Figure 4.4: Additional results for CRIBS

the radius evens out the differences between the initial parameter settings, allowing them all to perform reasonably well.

In the case of multimodal functions, adapting the distance is productive because it allows a slow-moving, nearly-converged particle to jump out of its current local minimum, facilitating search in other areas of the domain. Significantly, even CRIBS retains the ability to converge, but does so more slowly than CRS or baseline PSO.

Results for the benchmarks defined in Table 4.1 are found in Figure 4.4. DeJongF4, like Sphere, is smooth and unimodal. Griewank, while multimodal, begins to appear unimodal as the dimensionality increases. Ackley, SchafferF6, and SchafferF7 are highly multimodal and symmetric like Rastrigin; Rosenbrock is multimodal and asymmetric but appears unimodal when not in the region of the global minimum.

As expected, CRS and CRIBS are less effective on unimodal functions than baseline PSO. The Griewank function is interesting because it is unimodal until the proper level of detail is achieved, a fact that is evident in the slow initial drop but eventual good performance of CRIBS. With the possible exception of Rosenbrock, CRIBS works best on multimodal functions, and even on Rosenbrock it remains competitive.

Clearly, if it is known that the target function is smooth and unimodal, any kind of bouncing is a bad idea. When working with multimodal functions, however, using bouncing with both adaptive collision radius and bounce distance serves to improve performance while retaining reasonable convergence properties.

4.3 Attractive-Repulsive PSO

SEPSO is one of many diversity-increasing methods for PSO. The same authors later introduced the "Attractive and Repulsive PSO" (ARPSO), which uses a global

Table 4.1: Common benchmark functions

Ackley: $(-32.768, 32.768)$

$$f(\mathbf{x}) = 20 + e - 20\exp\left(\frac{-\|\mathbf{x}\|_2}{5\sqrt{D}}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right)$$

DeJongF4: $(-20, 20)$

$$f(\mathbf{x}) = \sum_{i=1}^{D} i x_i^4$$

Griewank: $(-600, 600)$

$$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Rastrigin: $(-5.12, 5.12)$

$$f(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 10\sum_{i=1}^{D} 1 - \cos(2\pi x_i)$$

Rosenbrock: $(-100, 100)$

$$f(\mathbf{x}) = \sum_{i=1}^{D-1} 100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2$$

SchafferF6: $(-100, 100)$

$$f(\mathbf{x}) = \frac{1}{2} + \frac{\sin^2\|\mathbf{x}\|_2 - \frac{1}{2}}{\left(1 + \frac{1}{1000}\|\mathbf{x}\|_2^2\right)^2}$$

SchafferF7: $(-100, 100)$

$$f(\mathbf{x}) = \sqrt{\|\mathbf{x}\|_2}\left(1 + \sin^2 50\sqrt[5]{\|\mathbf{x}\|_2}\right)$$

Sphere: $(-50, 50)$

$$f(\mathbf{x}) = \|\mathbf{x}\|_2^2$$

diversity metric to guide a swarm's exploratory behavior [Riget and Vesterstrøm 2002]. The diversity of a swarm S is

$$\operatorname{diversity}(S) = \frac{1}{|S|L} \sum_{i=1}^{|S|} \|\mathbf{x}_i - \bar{\mathbf{x}}\|_2 \tag{4.10}$$

which is essentially a measure of the average Euclidean distance of each particle from the center of mass:

$$\bar{\mathbf{x}} = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbf{x}_i \ . \tag{4.11}$$

Diversity is scaled by L, the length of the longest diagonal in the feasible region. The metric[1] is calculated globally at each iteration of PSO and is used to artificially inject diversity when needed, via "repulsion".

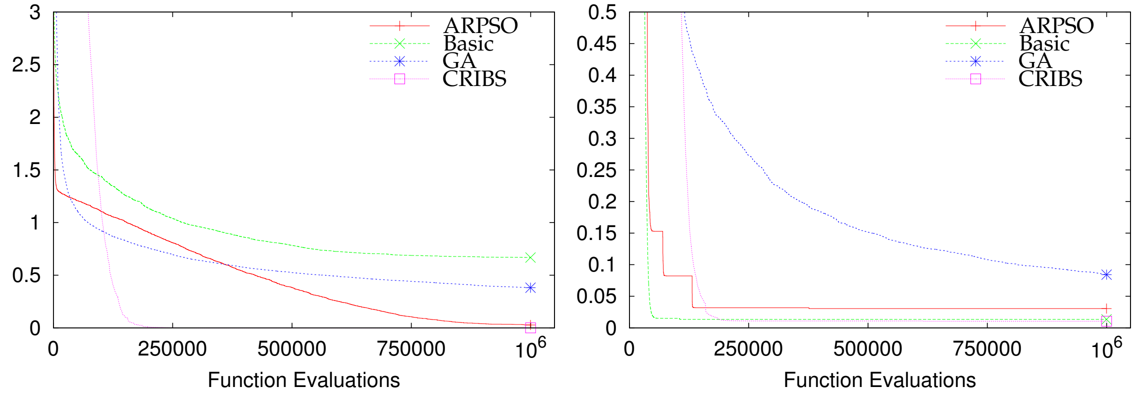
When the diversity falls below η^-, particles switch into repulsion mode; this is intended to make them fly away from each other, increasing diversity and allowing them to escape local minima. When the diversity has exceeded η^+, the particles are switched back to their normal behavior of attraction and the swarm begins once again to converge.

Attraction is the default behavior of PSO, so repulsion is achieved by adding a sign coefficient to the acceleration term in (4.2):

$$\dot{\mathbf{x}}_{t+1} = \chi \left(\dot{\mathbf{x}}_t + s \ddot{\mathbf{x}}_{t+1} \right) \ . \tag{4.12}$$

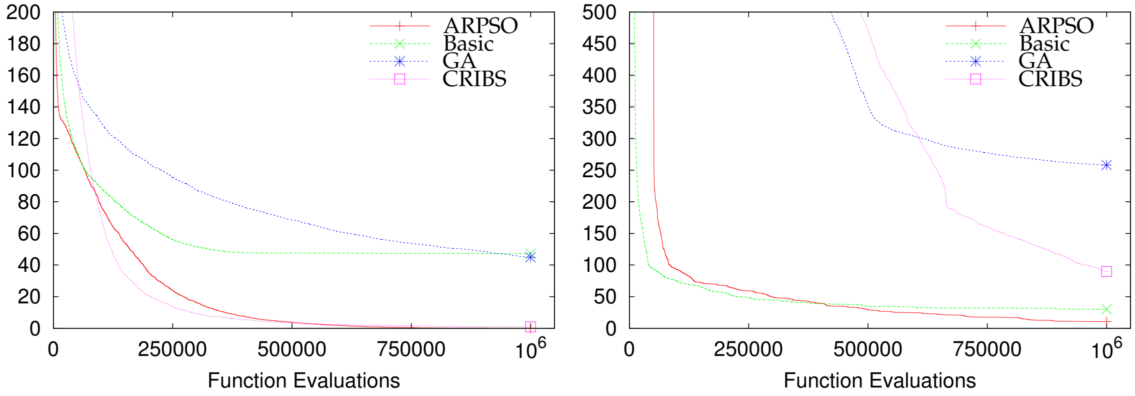
---

[1] The published definition uses **p** instead of **x** in (4.10) and (4.11), but this is unlikely to be correct: in order for diversity to increase, at least one particle must quickly find a better **p** while accelerating away from **g**, a highly unlikely event; implemented this way, ARPSO behavior is equivalent to that of standard PSO until entering repulsion mode, where it remains indefinitely without re-entering attraction mode or improving further over time.



(a) Ackley

(b) Griewank



(c) Rastrigin

(d) Rosenbrock

Figure 4.5: ARPSO compared with other techniques

Letting $s = -1$ switches the swarm into repulsion mode while $s = 1$ restores normal attraction behavior. Note that repulsion generally only has an effect on moving (less converged) particles.

Because ARPSO attempts to quantify the amount of clustering of the swarm in order to detect appropriate times to inject diversity, the use of Euclidean distance as the core of the diversity measure is not, in general, appropriate [Aggarwal et al. 2001]. This issue is less of a concern with SEPSO because it makes local decisions rather than detecting global clustering, but it nevertheless merits future study.

Figure 4.5 shows a direct comparison with the published ARPSO results for several 50-dimensional problems, averaged over 50 runs. The underlying algorithm for CRIBS is constricted PSO with default constriction parameters as described in the introduction, effectively eliminating problem-dependent parameter tuning. The initial radius and adaptation factor are in all cases set conservatively at $r = 0.5L$ and $\gamma = 0.9$, respectively. All other results are reproduced directly from the ARPSO paper, including the use of a linear scale instead of the log scale ubiquitously employed in this work; all that is known about the published ARPSO results is that a variant of constricted PSO was used as the underlying motion methodology, and problem-dependent parameters such as swarm size, and maximum velocity, and inertia weights were carefully tuned for each function [Riget and Vesterstrøm 2002].

The initial radius r and adaptation constant γ for the ARPSO comparisons in Figure 4.5 are intentionally different than those reported elsewhere in this work; the way that they were determined illustrates an important point: they were set conservatively (huge r, large γ, and standard constricted PSO) with no exploratory tuning. It was assumed that adaptation would adjust for any problems with the initial parameter settings, and it did. This characteristic of CRIBS makes its successful application to problems easy because it is robust to various parameter settings.

Even though ARPSO is reported to be using optimized parameter settings for each experiment, the untuned CRIBS performed at least as well on all problems but Rosenbrock, and on that benchmark CRIBS exhibits the same behavior that makes ARPSO itself attractive: it is avoiding premature convergence and is continuing on a downward trend.

It is natural to ask whether it is useful to adapt η^- and η^+ in the same way that r is adapted in CRS and CRIBS. While our efforts in this regard did result in some improvement, results were not consistent and even the improved versions of ARPSO failed to reliably outperform CRIBS.

4.4 Conclusions

Artificial diversity injection for a convergent algorithm like PSO is an interesting idea, but can require the manipulation of parameters that are nontrivial to tune. In addition, care must be taken to avoid eliminating desirable convergence that allows a swarm to explore the domain at decreasing scales, thereby gaining increasingly detailed information about a local minimum over time.

These issues are simultaneously addressed by allowing diversity to be injected less frequently as time progresses. In SEPSO, this is achieved by reducing the radius after every collision (CRS). The algorithm has the advantage of being simple to implement and more effective than its non-adaptive counterparts early in a run, especially on unimodal functions. Adapting the bounce distance (CRIBS) improves performance on multimodal functions while continuing to ensure eventual convergence. ARPSO does not appear to benefit from adaptation in the same way, and initial results suggest that CRIBS is a more robust approach in any case. Other diversity injection approaches (e.g. charged swarms [Blackwell and Bentley 2002]) may benefit from adaptation, an interesting topic for future research.

The adaptation of diversity parameters is simple to implement and has intuitive appeal, providing an effective way of increasing the exploration capabilities of PSO while retaining its desirable convergence properties.

Part II

Bayesian Modeling for Particle Swarm Optimization

Part II of this work consists of papers that consider PSO motion within a Bayesian framework. The paper in Chapter 5 introduces the use of the Kalman Filter as a means of determining an appropriate particle state for the next time step. This idea is important because it recasts the optimization problem as one of filtering and prediction; particle attractors form a trajectory as they move through the function domain, and the goal of the swarm becomes one of estimating and predicting the next step of that trajectory in order to find a new set of attractors.

Changing the problem in this way allows PSO to be thought of as a fundamentally Bayesian process: the Kalman Filter implies a Linear Dynamic System. The formulas involved have striking similarity to traditional PSO variants, suggesting a connection between the Kalman approach and more commonly used PSO methods. This connection is made evident in (though not directly established by) Chapter 6, which produces an algorithm similar to standard PSO by approximating the Kalman Filter.

This similarity, while not receiving much attention in the original paper, is explored in greater detail in Chapter 7. It begins by establishing the need for a more motivated approach to PSO algorithm design, and suggests that the need may be filled by starting with a model of the information relationships in swarm optimization. When solved using a Kalman Filter, one such model produces the algorithm in Chapter 5. Through a series of approximations the connection to traditional PSO becomes clear, providing new insights into why PSO works and how improvements may be made in a directed manner.

The last paper is important because it establishes the usefulness and significance of model-based algorithm design for swarm optimization; the model not only makes predictions about algorithm behavior, it also suggests places in which the algorithm may be improved. Because a model is used, the goals of optimization are clearly stated in the form of information relationships, and this explicit definition of intent is critical to a principled perspective on algorithm behavior.

The ideas in Chapter 7 prompted further thinking about Bayesian models for more general continuous optimization settings, providing the initial ideas for the final contribution of this work.

Chapter 5

The Kalman Swarm:
A New Approach to Particle Motion in Swarm Optimization

Published in Proceedings of GECCO 2004, Volume 1, pages 140–150

Abstract

Particle Swarm Optimization is gaining momentum as a simple and effective optimization technique. We present a new approach to PSO that significantly reduces the number of iterations required to reach good solutions. In contrast with much recent research, the focus of this work is on fundamental particle motion, making use of the Kalman Filter to update particle positions. This enhances exploration without hurting the ability to converge rapidly to good solutions.

5.1 Introduction

Particle Swarm Optimization (PSO) is an optimization technique inspired by social behavior observable in nature, such as flocks of birds and schools of fish [Kennedy and Eberhart 1995]. It is essentially a nonlinear programming technique suitable for optimizing functions with continuous domains (though some work has been done in discrete domains [Kennedy and Eberhart 1997]), and has a number of desirable properties, including simplicity of implementation, scalability in dimension, and good empirical performance. It has been compared to evolutionary algorithms such as GAs (both in methodology and performance) and has performed favorably [Kennedy and Spears 1998; Riget and Vesterstrøm 2002].

As an algorithm, it is an attractive choice for nonlinear programming because of the characteristics mentioned above. Even so, it is not without problems. PSO suffers from premature convergence, tending to get stuck in local minima [Riget and Vesterstrøm 2002; Løvbjerg 2002; Richards and Ventura 2003; Vesterstrøm et al. 2002]. We have also found that it suffers from an ineffective exploration strategy, especially around local minima, and thus does not find good solutions as quickly as it could. Moreover, adjusting the tunable parameters of PSO to obtain good performance can be a difficult task [Vesterstrøm et al. 2002; Shi and Eberhart 1998b].

Research addressing the shortcomings of PSO is ongoing and includes such changes as dynamic or exotic sociometries [Richards and Ventura 2003; Kennedy and Mendes 2002, 2003; Kennedy 1999; Mendes et al. 2003], spatially extended particles that bounce [Krink et al. 2002], increased particle diversity [Riget and Vesterstrøm 2002; Løvbjerg 2002], evolutionary selection mechanisms [Angeline 1998], and of course tunable parameters in the velocity update equations [Vesterstrøm et al. 2002; Shi and Eberhart 1998b; Clerc and Kennedy 2002]. Some work has been done that alters basic particle motion with some success, but the possibility for improvement in this area is still open [Kennedy 2003].

This paper presents an approach to particle motion that significantly speeds the search for optima while simultaneously improving on the premature convergence problems that often plague PSO. The algorithm presented here, KSwarm, bases its particle motion on Kalman filtering and prediction.

We compare the performance of KSwarm to that of the basic PSO model. In the next section, the basic PSO algorithm is reviewed, along with an instructive alternative formulation of PSO and a discussion of some of its shortcomings. Unless otherwise specified, "PSO" refers to the basic algorithm as presented in that section. Section 5.3 briefly describes Kalman Filters, and Section 5.4 describes KSwarm in detail. Experiments and their results are contained in Section 5.5. Finally, conclusions and future research are addressed in Section 5.6.

5.2 The Basic PSO Algorithm

PSO is an optimization strategy generally employed to find a global minimum. The basic PSO algorithm begins by scattering a number of "particles" in the function domain space. Each particle is essentially a data structure that keeps track of its current position $\vec{x}$ and its current velocity $\vec{v}$. Additionally, each particle remembers the "best" (lowest valued) position it has obtained in the past, denoted $\vec{p}$. The best of these values among all particles (the global best remembered position) is denoted $\vec{g}$.

At each time step, a particle updates its position and velocity by the following equations:

$$\vec{v}_{t+1} = \chi\Big(\vec{v}_t + \phi_1(\vec{p} - \vec{x}) + \phi_2(\vec{g} - \vec{x})\Big) \tag{5.1}$$

$$\vec{x}_{t+1} = \vec{x}_t + \vec{v}_{t+1} \ \ . \tag{5.2}$$

The *constriction coefficient* $\chi = 0.729844$ is due to Clerc and Kennedy [Clerc and Kennedy 2002] and serves to keep velocities from exploding. The stochastic scalars ϕ_1 and ϕ_2 are drawn from a uniform distribution over $[0, 2.05)$ at each time step. Though other coefficients have been proposed in an effort to improve the algorithm [Vesterstrøm et al. 2002; Shi and Eberhart 1998b], they will not be discussed here in detail.

5.2.1 An Alternative Motivation

Although the PSO update model initially evolved from simulated flocking and other natural social behaviors, it is instructive to consider an alternative motivation based on a randomized hill climbing search. A naive implementation may place a single particle in the function domain, then scatter a number of random sample points in the neighborhood, moving toward the best sample point at each new time step: $\vec{x}_{t+1} = \vec{g}_t$.

If the particle takes this step by first calculating a velocity, the position is still given by (5.2) and the velocity update is given by

$$\vec{v}_{t+1} = \vec{g}_t - \vec{x}_t \ \ . \tag{5.3}$$

As this type of search rapidly becomes trapped in local minima, it is useful to randomly overshoot or undershoot the actual new location in order to do some directed exploration (after all, the value of the new location is already known).

For similar reasons, it may be desirable to add momentum to the system, allowing particles to "roll out" of local minima. Choosing a suitable random scalar ϕ, this yields

$$\vec{v}_{t+1} = \vec{v}_t + \phi(\vec{g}_t - \vec{x}_t) \ . \tag{5.4}$$

The equation (5.4) is strikingly similar to (5.1). In fact, it is trivial to reformulate the PSO update equation to be of the same form as (5.1) [Clerc and Kennedy 2002; Mendes et al. 2003].

The fundamental difference between this approach and PSO is the way that $\vec{g}$ is calculated. In PSO, $\vec{g}$ is taken from other particles already in the system. In the approach described in this section, $\vec{g}$ is taken from disposable samples scattered in the neighborhood of a single particle.

This suggests that the basic PSO is a hill climber that uses existing information to reduce function evaluations. It is set apart more by its social aspects than by its motion characteristics, an insight supported by Kennedy but for different reasons [Kennedy 2003].

5.2.2 Particle Motion Issues

Given that PSO is closely related to an approach as simple as randomized hill climbing, it is no surprise that attempts to improve the velocity update equation with various scaling terms have met with marginal success. Instead, more fundamental changes such as increased swarm diversity, selection, and collision avoiding particles have shown the greatest promise [Riget and Vesterstrøm 2002; Løvbjerg 2002; Angeline 1998].

Unfortunately these methods are not without problems either, as they generally fail to reduce the iterations required to reach suitable minima. They focus primarily on eliminating stagnation, eventually finding better answers than the basic PSO without finding them any faster.

It has been pointed out that nonlinear programming is subject to a fundamental tradeoff between convergence speed and final fitness [Riget and Vesterstrøm 2002], suggesting that it is not generally possible to improve one without hurting the other. Fortunately, this tradeoff point has not yet been reached in the context of particle swarm optimization, as it is still possible to find good solutions more quickly without damaging final solution fitness.

For example, the development of a PSO visualization tool served to expose a particularly interesting inefficiency in the basic PSO algorithm. As the particles close in on $\vec{g}$ they tend to lose their lateral momentum very quickly, each settling into a simple periodic linear motion as they repeatedly overshoot (and undershoot) the target. This exploration strategy around local minima is very inefficient, suggesting that a change to particle motion may speed the search by improving exploration.

Such a change should ideally preserve the existing desirable characteristics of the algorithm. PSO is essentially a *social* algorithm, which gives it useful emergent behavior. Additionally, PSO motion is stochastic, allowing for randomized *exploration*. Particles also have *momentum*, adding direction to the random search. The constriction coefficient indicates a need for *stability*. Alterations to particle motion should presumably maintain these properties, making the Kalman Filter a suitable choice.

5.3 The Kalman Filter

Kalman filters involve taking noisy observations over time and using model information to estimate the true state of the environment [Kalman 1960]. Kalman filtering is generally applied to motion tracking problems. It may also be used for prediction by applying the system transition model to the filtered estimate.

The Kalman Filter is limited to normal noise distributions and linear transition and sensor functions and is therefore completely described by several constant matrices and vectors. Specifically, given an observation column vector $\mathbf{z}_{t+1}$, the Kalman Filter is used to generate a normal distribution over a belief about the true state. The parameters $\mathbf{m}_{t+1}$ and $\mathbf{V}_{t+1}$ of this multivariate distribution are determined by the following equations [Russel and Norvig 2003]:

$$\mathbf{m}_{t+1} = \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\mathbf{m}_t) \tag{5.5}$$

$$\mathbf{V}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x}) \tag{5.6}$$

$$\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x})\mathbf{H}^\top\Big(\mathbf{H}(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x})\mathbf{H}^\top + \mathbf{V}_\mathbf{z}\Big)^{-1} \ . \tag{5.7}$$

In these equations, $\mathbf{F}$ and $\mathbf{V}_\mathbf{x}$ describe the system transition model while $\mathbf{H}$ and $\mathbf{V}_\mathbf{z}$ describe the sensor model. The equations require a starting point for the filtered belief, represented by a normal distribution with parameters $\mathbf{m}_0$ and $\mathbf{V}_0$, which must be provided.

The filtered or "true" state is then represented by a distribution:

$$\mathbf{x}_t \sim \mathrm{Normal}(\mathbf{m}_t, \mathbf{V}_t) \ . \tag{5.8}$$

This distribution may be used in more than one way. In some applications, the mean $\mathbf{m}_t$ is assumed to be the true value. In others, the distribution is sampled once to obtain the value. In this work, the latter is done.

The above describes how to do Kalman *filtering*, yielding $\mathbf{m}_t$ from an observation $\mathbf{z}_t$. A simple form of *prediction* involves applying the transition model to obtain a belief about the next state $\mathbf{m}'_{t+1}$:

$$\mathbf{m}'_{t+1} = \mathbf{F}\mathbf{m}_t \ . \tag{5.9}$$

There are other forms of prediction, but this simple approach is sufficient for the introduction of the algorithm in the next section, and for its use in particle swarms.

5.4 The Kalman Swarm (KSwarm)

KSwarm defines particle motion entirely from Kalman prediction. Each particle keeps track of its own $\mathbf{m}_t$, $\mathbf{V}_t$, and $\mathbf{K}_t$. The particle then generates an observation for the Kalman filter with the following formulae:

$$\vec{z}_v = \phi(\vec{g} - \vec{x}) \tag{5.10}$$

$$\vec{z}_p = \vec{x} + \vec{z}_v \ . \tag{5.11}$$

Similar to PSO, ϕ is drawn uniformly from $[0, 2)$, and the results are row vectors. The full observation vector is given by making a column vector out of the concatenated position and velocity row vectors: $\mathbf{z} = (\vec{z}_p, \vec{z}_v)^\top$. This observation is then used to generate $\mathbf{m}_{t+1}$ and $\mathbf{V}_{t+1}$ using (5.5), (5.6), and (5.7)

Once the filtered value is obtained, a prediction $\mathbf{m}'_{t+2}$ is generated using (5.9). Together, $\mathbf{m}'_{t+2}$ and $\mathbf{V}_{t+1}$ parameterize a normal distribution. We say, then, that

$$\mathbf{x}_{t+1} \sim \text{Normal}(\mathbf{m}'_{t+2}, \mathbf{V}_{t+1}) \ . \tag{5.12}$$

The new state of the particle is obtained by sampling once from this distribution. The position of the particle may be obtained from the first half of $\mathbf{x}^\top_{t+1}$, and the velocity (found in the remaining half) is unused.

This method for generating new particle positions has at least one immediately obvious advantage over the original approach: there is no need for a constriction coefficient. Particle momentum comes from the state maintained by the Kalman Filter rather than from the transition model. In our experiments, this eliminated the need for any explicit consideration of velocity explosion.

Table 5.1: Domains of Test Functions

Function	Domain
Sphere	$(-50, 50)^d$
DeJongF4	$(-20, 20)^d$
Rosenbrock	$(-100, 100)^d$
Griewank	$(-600, 600)^d$
Rastrigin	$(-5.12, 5.12)^d$

5.5 Experiments

KSwarm was compared to PSO in five common test functions: Sphere, DejongF4, Rosenbrock, Griewank, and Rastrigin. The first three are unimodal while the last two are multimodal. In all experiments, the dimensionality $d = 30$. The definitions of the five functions are given here:

$$\text{Sphere}(\vec{x}) = \sum_{i=1}^{d} x_i^2 \tag{5.13}$$

$$\text{DeJongF4}(\vec{x}) = \sum_{i=1}^{d} i x_i^4 \tag{5.14}$$

$$\text{Rosenbrock}(\vec{x}) = \sum_{i=1}^{d-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \tag{5.15}$$

$$\text{Rastrigin}(\vec{x}) = \sum_{i=1}^{d} x_i^2 + 10 - 10\cos(2\pi x_i) \tag{5.16}$$

$$\text{Griewank}(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \ . \tag{5.17}$$

The domains of these functions are given in Table 5.1.

5.5.1 Experimental Parameters

In all experiments, a swarm size of 20 was used. Though various sociometries are available, the *star* (or *gbest* [Kennedy and Eberhart 1995]) sociometry was used almost exclusively in the experiments because it allows for maximum information flow [Krink et al. 2002]. Each experiment was run 50 times for 1000 iterations, and the results were averaged to account for stochastic differences. The parameters to (5.5), (5.6), and (5.7) are given below, and are dependent on the domain size of the function. The vector containing the size of the domain in each dimension is denoted $\vec{w}$. The column vector $\mathbf{w} = (\vec{w}, \vec{w})^\top$ is formed from two concatenated copies of $\vec{w}$. In the following equations, $\mathbf{I}_n$ is an identity matrix with n rows.

$$\mathbf{m}_0 = 0 \qquad \mathbf{V}_0 = \theta \operatorname{diag}(\mathbf{w}) \tag{5.18}$$

$$\mathbf{H} = \mathbf{I}_{2d} \qquad \mathbf{V}_\mathbf{z} = \theta \operatorname{diag}(\mathbf{w}) \tag{5.19}$$

$$\mathbf{F} = \begin{pmatrix} \mathbf{I}_d & \mathbf{I}_d \\ \mathbf{0} & \mathbf{I}_d \end{pmatrix} \qquad \mathbf{V}_\mathbf{x} = \theta \operatorname{diag}(\mathbf{w}) \ . \tag{5.20}$$

The initial mean $\mathbf{m}_0$ is a column vector of $2d$ zeros. The scalar θ indicates how large the variance should be in each dimension, and was set to 0.0001 for all experiments, as this produced a variance that seemed reasonable. The transition function simply increments position by velocity while leaving the velocity untouched.

All of the vectors used in the Kalman equations are of length $2d$ and all matrices are square and of size $2d$. This is the case because the model makes use of velocity as well as position, so extra dimensions are needed to maintain and calculate the velocity as part of the state. This implies that the sample obtained from (5.12) is *also* a vector of length $2d$, the first half of which contains position information. That position information is used to set the new position of the particle and the velocity information is unused except for the next iteration of the Kalman update equations.

Table 5.2: PSO vs. KSwarm Final Values

Function	PSO	KSwarm
Sphere	370.041	**4.723**
DejongF4	4346.714	**4.609**
Rosenbrock	2.61e7	**3.28e3**
Griewank	13.865	**0.996**
Rastrigin	106.550	**53.293**

5.5.2 Results

Table 5.2 shows the final values reached by each algorithm after 1000 iterations were performed. It is clear from the table that the KSwarm obtains values that are often several orders of magnitude better than the original PSO algorithm.

In addition to obtaining better values, the KSwarm tends to find good solutions in fewer iterations than the PSO, as evidenced by Figs. 5.1, 5.2, 5.3, 5.4, and 5.5. Note that each figure has a different scale.

Because the results obtained using the star sociometry were so striking, this experiment was also run using a sociometry where each particle had 5 neighbors. The corresponding results were so similar as to not warrant inclusion in this work.

These results represent a clear and substantial improvement over the basic PSO, not only in the final solutions, but in the speed with which they are found. It should be noted that much research has been done to improve PSO in other ways and that KSwarm performance in comparison to these methods has not been fully explored. The purpose of this work is to demonstrate a novel approach to particle motion that substantially improves the basic algorithm. The comparison and potential combination of KSwarm with other PSO improvements is part of ongoing research and will be a subject of future work.

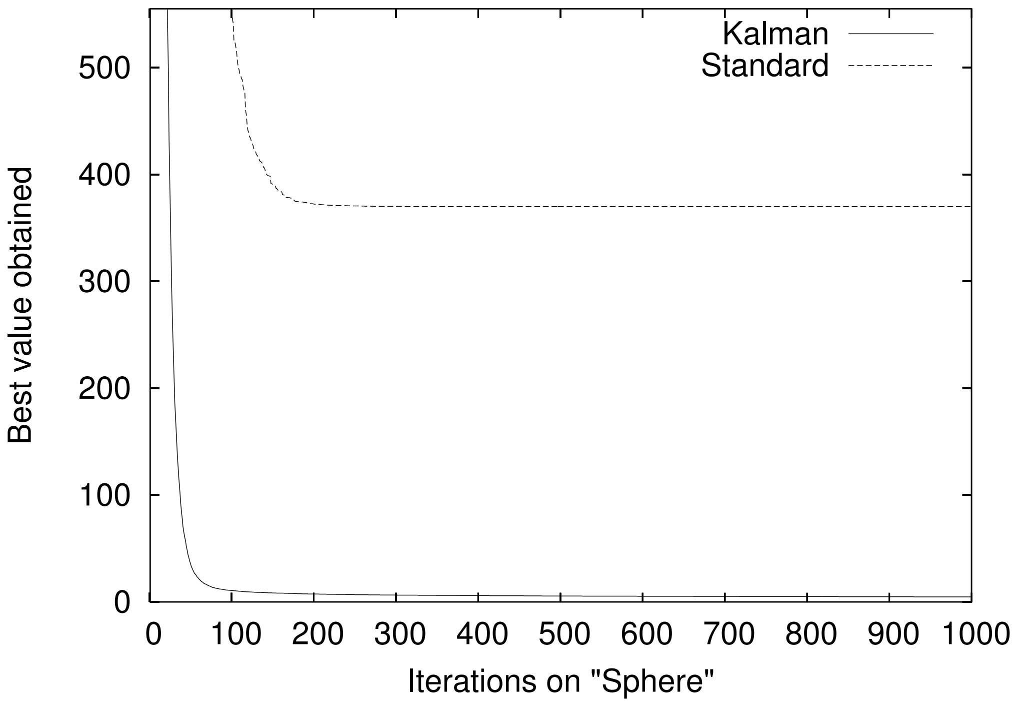


Figure 5.1: Sphere

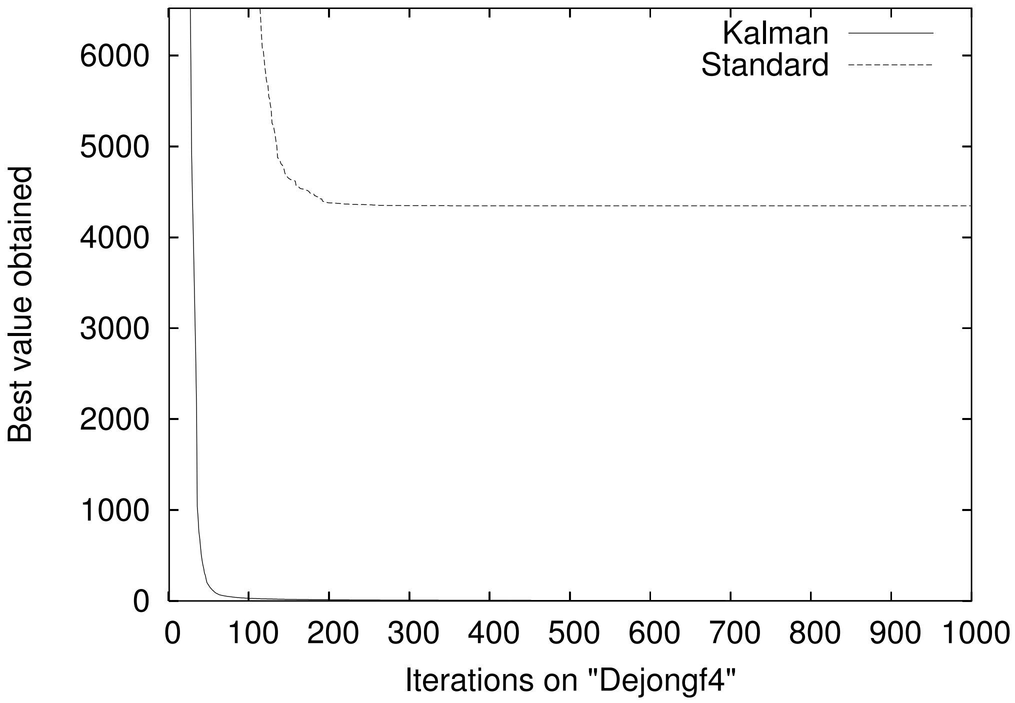


Figure 5.2: DeJongF4

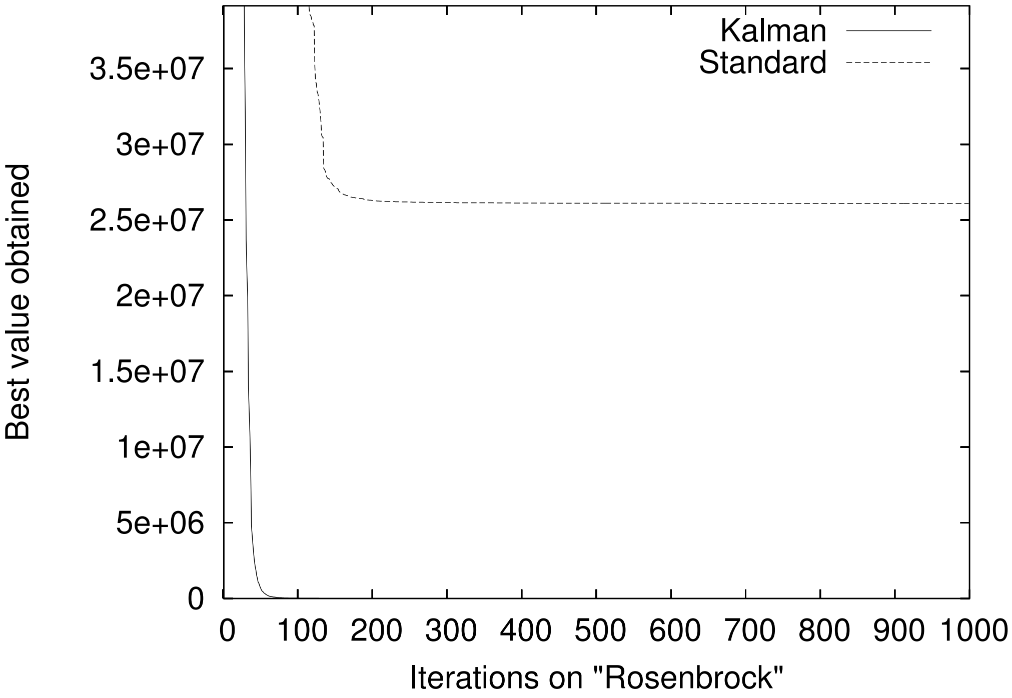


Figure 5.3: Rosenbrock

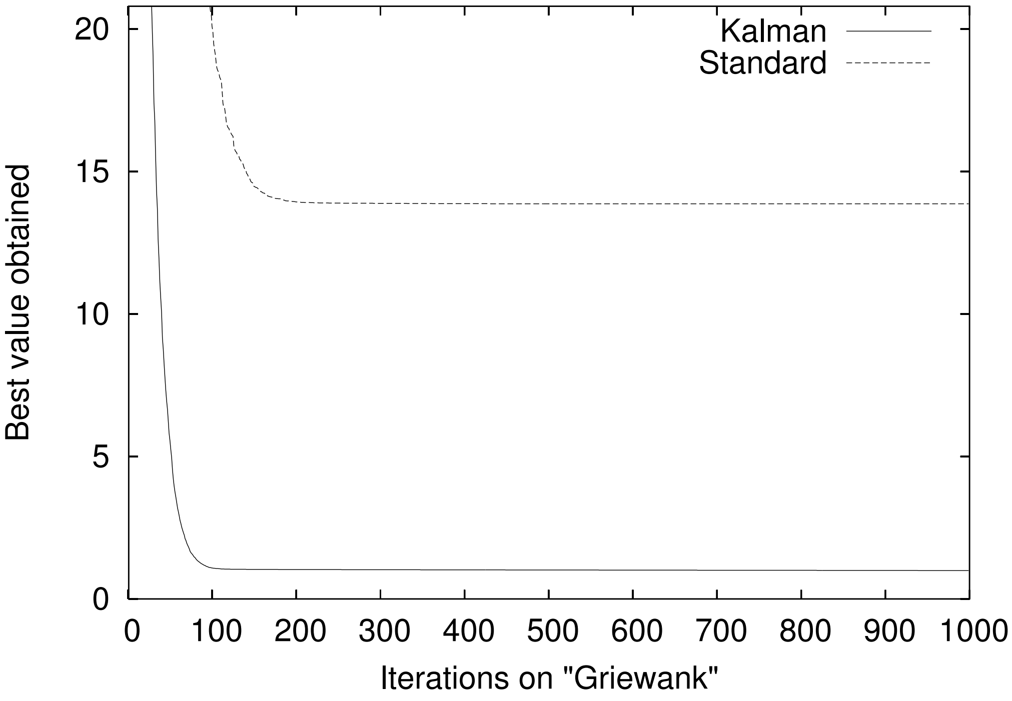


Figure 5.4: Griewank

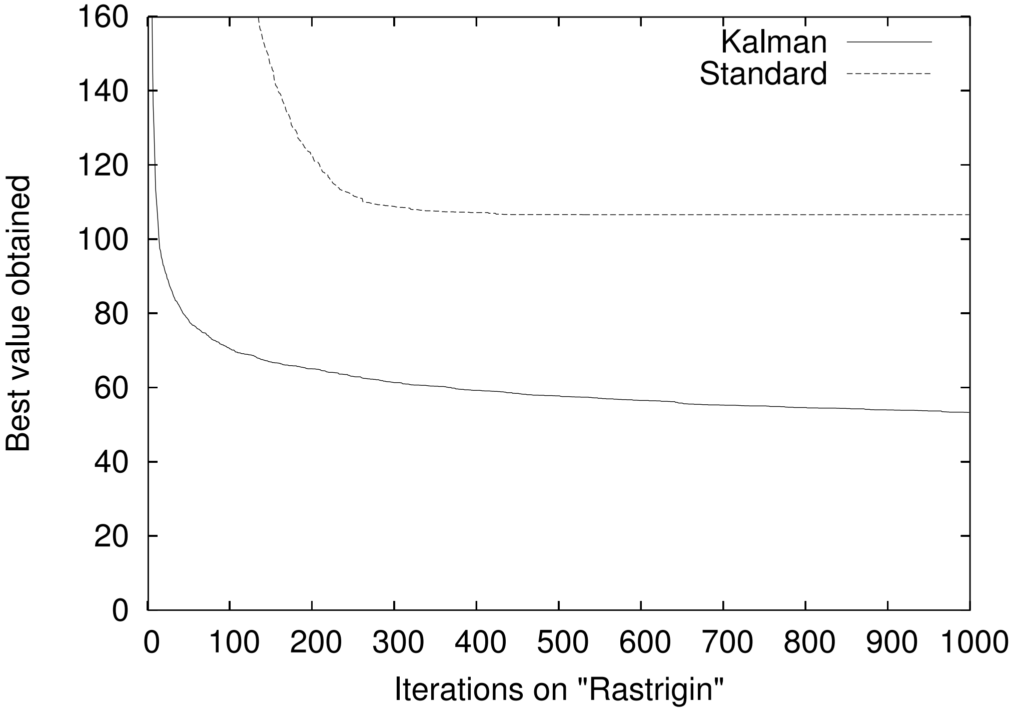


Figure 5.5: Rastrigin

5.5.3 Notes on Complexity

It is worth noting that the Kalman motion update equations require more computational resources than the original particle motion equations. In fact, because of the matrix operations, the complexity is $O(d^3)$ in the number of dimensions ($d = 30$ in our experiments). The importance of this increased complexity, however, appears to diminish when compared to the apparent exponential improvement in the number of iterations required by the algorithm. Additionally, the complexity can be drastically reduced by using matrices that are mostly diagonal or by approximating the essential characteristics of Kalman behavior in a simpler way.

5.6 Conclusions and Future Work

It remains to be seen how KSwarm performs against diversity-increasing approaches, but preliminary work indicates that it will do well in that arena, especially with regard to convergence speed. Since many methods which increase diversity do not fundamentally change particle motion update equations, combining this approach with those methods is simple. It can allow KSwarm to not only find solutions faster, but also to avoid the stagnation to which it is still prone.

Work remains to be done on alternative system transition matrices. The transition model chosen for the motion presented in this work is not the only possible model; other models may produce useful behaviors. Additionally, the complexity of the algorithm should be addressed. It is likely to be easy to improve by simple optimization of matrix manipulation, taking advantage of the simplicity of the model. More work remains to be done in this area.

KSwarm fundamentally changes particle motion as outlined in PSO while retaining its key properties of sociality, momentum, exploration, and stability. It represents a substantial improvement over the basic algorithm not only in the resulting solutions, but also in the speed with which they are found.

Chapter 6

Improving on the Kalman Swarm: Extracting Its Essential Characteristics

Published in Proceedings of GECCO 2004, Late-Breaking Papers

Abstract

The Kalman Swarm (KSwarm) is a new approach to particle motion in PSO that reduces the number of iterations required to reach good solutions. Unfortunately, it has much higher computational complexity than basic PSO. This paper addresses the runtime of KSwarm in a new algorithm called "Linear Kalman Swarm" (LinkSwarm) which has linear complexity and performs even better than KSwarm. Some possible reasons for the success of KSwarm are also explored.

6.1 Introduction

The Kalman Swarm (KSwarm) [Monson and Seppi 2004] is an adaptation of standard PSO [Kennedy 1999] that uses the Kalman Filter [Kalman 1960] to calculate the next position of each particle. The best location in its neighborhood (which always includes the particle itself) is used as an observation at each time step, producing a new location through prediction. This approach has been shown to produce better solutions in fewer iterations than standard PSO in a variety of test functions.

KSwarm, however, is not without liabilities. The dimensional complexity of running the algorithm is O(d^3), which is much higher than the O(d) complexity of basic PSO. Furthermore, KSwarm has a number of input parameters that are difficult to tune.

This paper presents the new algorithm "Linear Kalman Swarm" (LinkSwarm), which is an approximation to KSwarm with linear complexity that obtains better performance. The algorithm is developed by analyzing and extracting the essential characteristics of KSwarm. Through its development, some ideas are generated that may help explain the good performance of KSwarm. Results that compare the performance of the two algorithms are given on various test functions.

6.2 KSwarm Speed

Although KSwarm can often get better results in fewer iterations than basic PSO, each iteration is far more expensive. Because the complexity of KSwarm is O(d^3), this significantly increases the running time of the algorithm and is especially a problem when optimizing high-dimensional functions.

We will therefore attempt to extract the essential characteristics of the KSwarm algorithm in order to provide a fast approximation. The Kalman Filter is the basis for the update equations, and these will be analyzed first. Recall that $\mathbf{F}$ and $\mathbf{H}$ are the transition and sensor characteristic matrices with $\mathbf{V}_\mathbf{x}$ and $\mathbf{V}_\mathbf{z}$ as their respective covariance matrices, $\mathbf{m}_t$ and $\mathbf{V}_t$ are the mean and covariance parameters of the filtered state at time t, and $\mathbf{K}_t$ is the "Kalman Gain" at time t. These parameters and the following update equations comprise the multivariate Kalman Filter:

$$\mathbf{m}_{t+1} = \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\mathbf{m}_t) \tag{6.1}$$

$$\mathbf{V}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x}) \tag{6.2}$$

$$\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x})\mathbf{H}^\top\Big(\mathbf{H}(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V}_\mathbf{x})\mathbf{H}^\top + \mathbf{V}_\mathbf{z}\Big)^{-1} \quad . \tag{6.3}$$

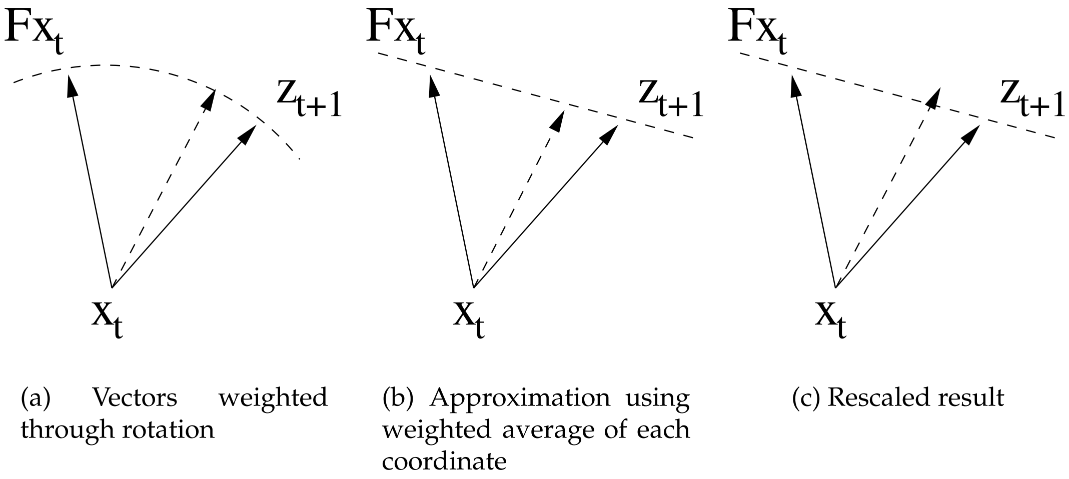
The complexity of KSwarm is due almost entirely to complex matrix operations, like multiplication and inversion. This complexity is present in the update equations above and in sampling from a multivariate Normal distribution. Any approximation intended to speed up the algorithm should address these two key areas.

6.2.1 Weighted Vectors

Because each particle knows with certainty where its neighbors are, we may assume that the system has perfect sensors ($\mathbf{H} = \mathbf{I}$), and may rearrange (6.1) thus:

$$\begin{aligned}\mathbf{m}_{t+1} &= \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{F}\mathbf{m}_t) \\ &= \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}\mathbf{z}_{t+1} - \mathbf{K}_{t+1}\mathbf{F}\mathbf{m}_t \\ &= (\mathbf{I} - \mathbf{K}_{t+1})\mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}\mathbf{z}_{t+1} \quad .\end{aligned} \tag{6.4}$$

This makes the filtered state look like a convex combination (assuming, of course, that $\mathbf{K}_{t+1}$ has the appropriate properties) of the predicted next state $\mathbf{F}\mathbf{m}_t$ and the observed state $\mathbf{z}_{t+1}$. It seems reasonable to assume that this is an essential characteristic of the Kalman Filter: it balances observations and myopic predictions. If



(a) Vectors weighted through rotation

(b) Approximation using weighted average of each coordinate

(c) Rescaled result

Figure 6.1: Weighting two vectors to generate a third

this is indeed a governing principle of a Kalman Filter, then it should be possible to construct a useful approximation of that behavior.

Since the Kalman Filter is basically balancing between myopic predictions and neighborhood observations using the linear operator **K**, we may view this as generating a vector through rotation and scaling of one of the two other vectors. This operation may be done by rotating one of the vectors by a specified angle through the plane defined by both vectors (a matrix operation that we would like to avoid), or it may be approximated by taking a weighted average of the normalized vectors and performing some post scaling. The two approaches are depicted in Figure 6.1.

As the weighted average is a much cheaper computation than rotation, this is what is done. Though it does not produce precisely the same results when the new vector is very close to one of the original vectors, it does provide a fast and useful approximation. To make the weighted average work correctly, we first normalize the original vectors, take a weighted sum to generate a new vector, normalize it, and then scale it so that its length is a weighted sum of the original lengths.

More formally, if we have a weight scalar $\alpha \in [0, 1]$ and vectors $\mathbf{u_1}$ and $\mathbf{u_2}$, we may generate a new vector $\mathbf{v}$ thus:

$$\widehat{\mathbf{v}} = \alpha \frac{\mathbf{u_1}}{||\mathbf{u_1}||} + (1 - \alpha) \frac{\mathbf{u_2}}{||\mathbf{u_2}||} \tag{6.5}$$

$$\mathbf{v} = \frac{\widehat{\mathbf{v}}}{||\widehat{\mathbf{v}}||} \left(\alpha ||\mathbf{u_1}|| + (1 - \alpha) ||\mathbf{u_2}|| \right) \quad . \tag{6.6}$$

Care must be taken when $\mathbf{u_1}$ and $\mathbf{u_2}$ point in opposite directions. In this case we have two options: point the new vector in some direction orthogonal to one of the vectors (of which there are two if $d = 2$ and infinite if $d > 2$) or pick some other direction. In practice this is solved by simply picking one of the original vectors as the new vector, usually that which has a better associated value. This case is easily detected by calculating $\widehat{\mathbf{v}}$ using (6.5) and then noting that $||\widehat{\mathbf{v}}||$ is extremely small.

The behavior of KSwarm is approximated using this approach to calculate a new velocity for a particle. The particle's current velocity is balanced against the relative best position from its neighborhood. The new velocity is some vector between the two, based on α.

We were previously using $\mathbf{m}_t$ as the current position and velocity of the particle, but now we compute only the velocity using the approximation outlined above. We add the velocity to our current position to get a new position. Therefore, there is no longer any need to compute $\mathbf{m}_t$ nor is there any use for $\mathbf{F}$, as the prediction of our next position is trivially calculated using the current velocity. It is worth noting that though this is beginning to sound like the original PSO algorithm, it is not (see Section 6.2.4).

6.2.2 Sampling From the Normal

So far, the approximation contains no randomness at all, effectively removing the complexity introduced from generating samples from a multivariate Normal.

Randomness, however, is an essential part of the original algorithm, so some sampling should be done to determine the final velocity of the particle.

KSwarm particles each sampled their final state from a multivariate Normal with parameters $\mathbf{Fm}_t$ and $\mathbf{V}_t$. They were provided with diagonal matrices for $\mathbf{V}_{t=0}$, $\mathbf{V}_\mathbf{x}$, and $\mathbf{V}_\mathbf{z}$. If $\mathbf{V}_t$ were to remain diagonal, we could optimize the multivariate sample (which involves matrix operations) with d univariate samples.

Therefore, instead of a covariance matrix, we select a variance vector $\boldsymbol{\sigma}^\mathbf{2}$ for our approximate algorithm. We further assume that this vector is constant (unlike $\mathbf{V}_t$ in the Kalman Filter). Had $\mathbf{V}_t$ been constant in the Kalman Filter, the gain $\mathbf{K}_t$ would have also been constant, as $\mathbf{V}_t$ was the only time-dependent portion of (6.3). This assumption of a constant $\boldsymbol{\sigma}^\mathbf{2}$ allows us to further assume that α is constant.

To generate randomness, the approximate algorithm uses each component of the normalized final velocity vector $\widehat{\frac{\mathbf{v}}{\|\mathbf{v}\|}}$ as the mean of a distribution. The variance is given by the corresponding component of $\boldsymbol{\sigma}^\mathbf{2}$. Since we are dealing with a normalized mean, it is not unreasonable to assume that $\boldsymbol{\sigma}^\mathbf{2} = \sigma^2\mathbf{e}$, where σ^2 is a constant scalar (and $\mathbf{e}$ is a vector of all ones). This effectively reduces the number of tunable parameters and simplifies the algorithm even further.

6.2.3 Linear Kalman Swarm (LinkSwarm)

The two fundamental approximations above complete the development of LinkSwarm, an approximation to KSwarm. The full LinkSwarm algorithm for particle motion is summarized in Algorithm 2.

The performance and runtimes of LinkSwarm and KSwarm are shown in Figure 6.2. For all experiments, $d = 30$, $\sigma^2 = 0.6$, $\alpha = 0.45$, the sociometry is "star", and 20 particles are used. Also, KSwarm uses a prior based on initial particle positions.

Algorithm 2 LinkSwarm

Given a particle's current velocity $\mathbf{v_t}$ and the *relative* position of the neighborhood best $\mathbf{v}_{\mathrm{best}}$ (the particle is always part of its own neighborhood in LinkSwarm), generate a new normal velocity vector $\mathbf{v}'$ as follows (norm generates a normalized vector):

$$\mathbf{v}' = \mathrm{norm}\left(\alpha \frac{\mathbf{v_t}}{||\mathbf{v_t}||} + (1-\alpha)\frac{\mathbf{v}_{\mathrm{best}}}{||\mathbf{v}_{\mathrm{best}}||}\right) \ . \tag{6.7}$$

For each coordinate v'_i of $\mathbf{v}'$, generate $\hat{\mathbf{v}}$ by drawing samples from a Normal distribution:

$$\hat{v}_i \sim \mathrm{Normal}(v'_i, \sigma^2) \ . \tag{6.8}$$

Normalize $\hat{\mathbf{v}}$ and scale to a weighted sum of the lengths of $\mathbf{v_t}$ and $\mathbf{v}_{\mathrm{best}}$ to create $\mathbf{v_{t+1}}$:

$$\mathbf{v_{t+1}} = (\alpha||\mathbf{v_t}|| + (1-\alpha)||\mathbf{v}_{\mathrm{best}}||)\,\mathrm{norm}(\hat{\mathbf{v}}) \ . \tag{6.9}$$

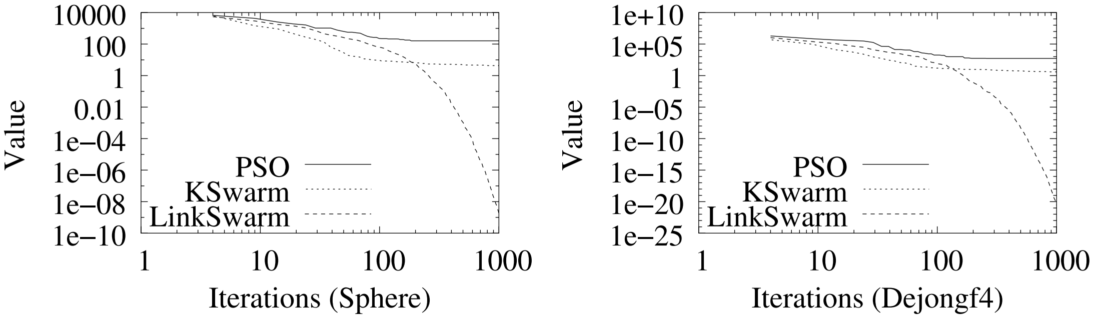
Add this velocity to the current position $\mathbf{x_t}$ to get $\mathbf{x_{t+1}}$.

It is interesting that LinkSwarm, which runs roughly 10 times faster than the original KSwarm in 30 dimensions, also performs better on every test function. Figure 6.2(f) also shows that the approximate algorithm is only very slightly slower than basic PSO (because of magnitude calculations and multiple Normal samples) but, in contrast with KSwarm, has the same O(d) computational complexity.

It appears that we have not only maintained the essential characteristics of KSwarm, but that we have improved on it in the process.

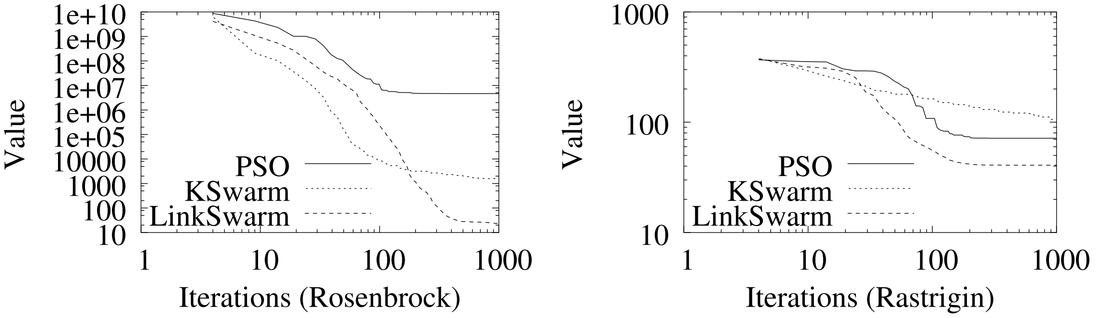
6.2.4 LinkSwarm vs. PSO

With all of the approximations made, the algorithm has been reduced to (roughly) a weighted sum of two vectors. Arguably, PSO does the same thing. The question arises, then, as to whether we really do anything different from basic PSO, besides sampling from a Normal distribution to get the final result. The answer is "yes".



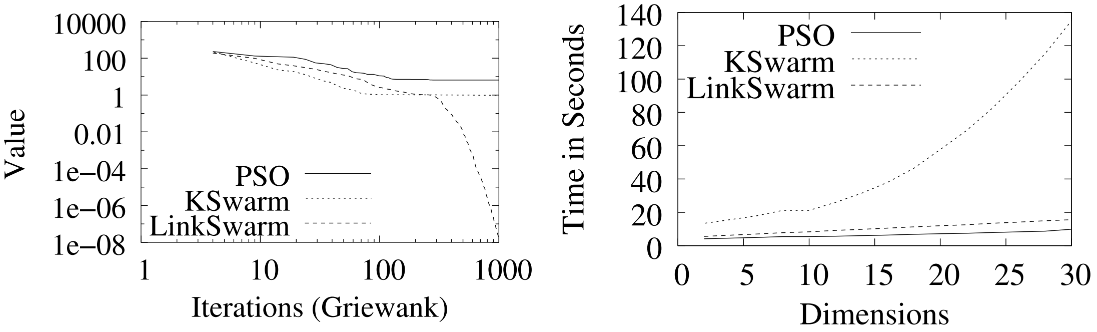
(a) Sphere

(b) DejongF4



(c) Rosenbrock

(d) Rastrigin



(e) Griewank

(f) Average Runtimes on Sphere

Figure 6.2: Performance of LinkSwarm as compared with original KSwarm

Basic PSO and many of its variants also take a “weighted sum” of two vectors in order to generate a new velocity. There are two very important differences, however, between LinkSwarm and basic PSO: the vectors involved in the weighting operation are different, and the velocity is *set* to a new value, not *augmented* by it. In other words, LinkSwarm generates a velocity, not an acceleration.

The fact that LinkSwarm sets rather than augments its current velocity neatly sidesteps the issue of velocity explosion. Of more interest, however, is the fact that it uses different vectors in its decision process. In basic PSO, the two vectors involved in the weighting operation are the relative position of the particle’s personal best and the relative position of the neighborhood best. LinkSwarm uses a particle’s *current velocity* (which may also be viewed as the relative predicted position at the next time step) instead of the relative position of its personal best. It only uses its personal best in the context of its neighborhood.

This basic difference between KSwarm and PSO is explored further in Section 6.4 as part of the explanation for its success.

6.3 KSwarm Deficiencies

When compared with the basic PSO algorithm, KSwarm performs very well, reducing the number of iterations required to reach good solutions and improving on the overall solutions on a number of test functions. This is an interesting result, as it does so by altering the motion characteristics of particles. Fundamental changes in motion have not received much attention in recent research because motion has been considered to be less important than the social and diversity aspects of the algorithm [Kennedy 2003; Kennedy and Mendes 2002; Riget and Vesterstrøm 2002; Richards and Ventura 2003; Mendes et al. 2003; Kennedy and Mendes 2003; Angeline 1998; Kennedy 1999, 2000; Vesterstrøm et al. 2002; Shi and Eberhart 1999;

Krink et al. 2002; Shi and Eberhart 1998b]. KSwarm represents a possible counterexample to that often implicit assumption.

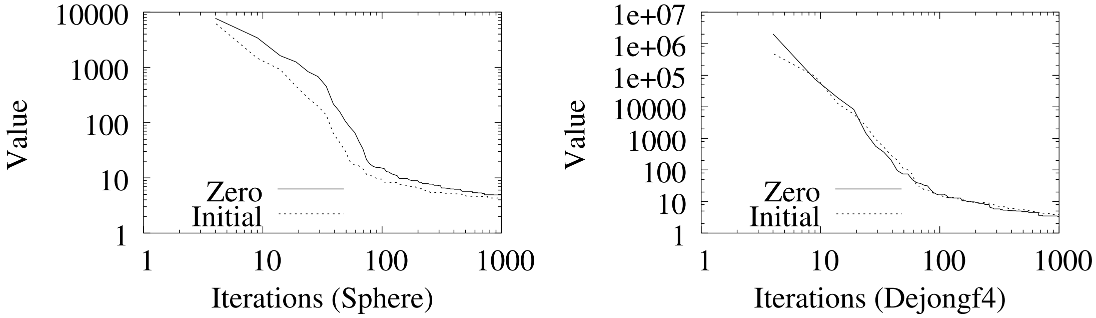
One may ask why Kalman Filter motion works well in this context. There is one rather unflattering reason that it works especially well on the test functions: the original KSwarm is subtly origin-seeking. The Kalman Filter requires a prior distribution on the particle's state. Recall that the state of a particle has been defined to be the position and velocity of that particle [Monson and Seppi 2004]. Absent a reasonable prior, this was always set to $\mathbf{0}$, indicating that the particle began its life with the assumption that it was at the origin and that it was not moving. In other words, it assumed it was *already at the optimum*, a deficiency addressed here.

A change was made to KSwarm, setting the prior of each particle to its initial position instead of $\mathbf{0}$. The graphs in Figure 6.3 compare the two approaches. It is clear that especially for strongly multi-modal functions like Rastrigin, the prior has a significant impact on the algorithm's behavior.

While this change significantly degrades the performance of KSwarm on Rastrigin, it leaves it practically unchanged on the other test functions, most of which are unimodal. Griewank is of particular interest because it is multimodal but the performance did not degrade for that function. Some attempt to explain this behavior is given in the next section.

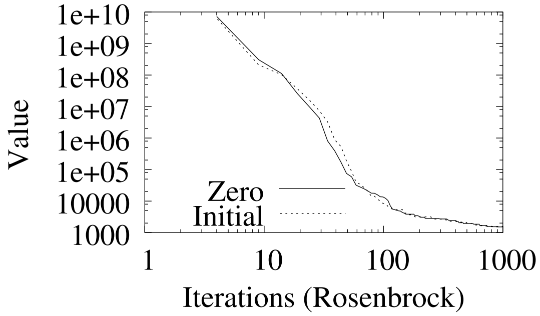
6.4 Why It Works

It is first important to acknowledge that the basic PSO algorithm has had many changes applied in recent years that substantially improve its performance. These methods are not considered here due to space and time constraints, but it is worth noting that LinkSwarm outperforms many of them where KSwarm did not. The results of those experiments are reserved for a later, more complete paper.

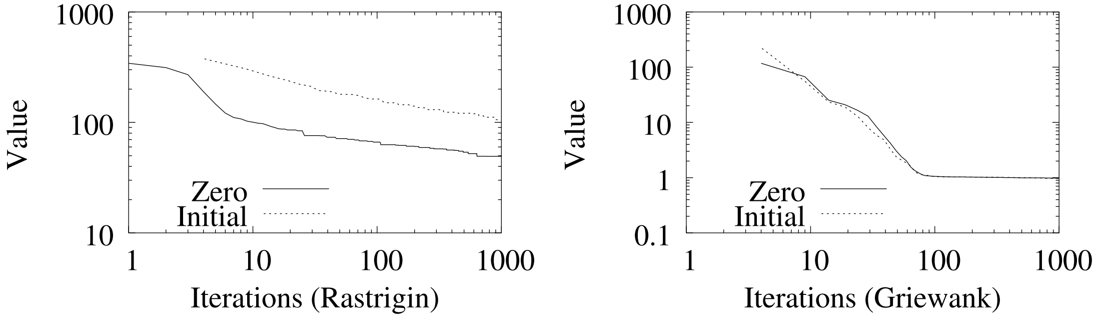


(a) Sphere

(b) DejongF4



(c) Rosenbrock



(d) Rastrigin

(e) Griewank

Figure 6.3: Results of KSwarm with a zero prior and with a more reasonable prior based on initial particle positions

The development of LinkSwarm served to expose some interesting characteristics of the original KSwarm. The most striking of these has already been mentioned, and is best understood in contrast to basic PSO: the vectors involved in the decision process are different. While basic PSO particles use personal and neighborhood bests to make their decisions, LinkSwarm particles throw their personal best in with their neighbors and decide between the neighborhood best and their current trajectory. This difference represents an interesting and useful assumption: a particle's current direction is probably pretty good.

The validity of this assumption depends on the kinds of functions that are being optimized. It is motivated by the fact that the particle was already approaching a good area to begin with, so there is no reason that it should have to stop and turn around just because one of its neighbors wants it to see something; it may be onto something good already.

This assumption works especially well with functions that, though potentially multimodal, have an overall trend. Functions like Rastrigin and Griewank fit this criterion quite nicely. Though they have many local minima, they each exhibit an overall downward trend toward the global minimum. A particle that is able to filter out the "noise" of the local minima in favor of discovering the overall trend is in general going to do well with these types of functions. The Kalman Filter was designed to do precisely that. These particles are expected to do very well on functions where sufficient "blurring" will produce a unimodal function.

6.5 Conclusions and Future Research

An enormous number of things have yet to be tried. Currently the weight parameter α is constant, but it could be changed dynamically, perhaps based on some confidence parameter that is learned over time. This needs to be explored more fully. The effect of the various parameters in general is not currently known, and that is also begging for more exploration.

These algorithms need to be tested against the state of the art. The basic PSO algorithm is known to be naive and suboptimal, and much research has been done to improve it. In order to be truly interesting, LinkSwarm and its variants must be compared with the best PSO algorithms known to date. Preliminary experiments show that it compares very favorably with recent PSO improvements.

Surprisingly, the approximate algorithm performed better than the original. The reasons for this are unknown, though we have some preliminary ideas. The parameters used to initialize KSwarm might not have been optimal, causing it to perform somewhat more poorly than it might otherwise have done. While this could potentially be explored to good effect, there are simply too many parameters to make such exploration feasible. The number of tunable parameters in KSwarm was one of its major shortcomings. The LinkSwarm algorithm not only has fewer and more intuitive parameters, but it is faster and works better.

LinkSwarm is a major improvement over KSwarm, not only in computational complexity, but also in the results obtained. It represents the elimination of numerous tunable parameters and is much simpler to code and to understand. Along with its improved speed and simplicity, preliminary work suggests that it does better than many recent improvements to PSO, potentially making it a strong contender in the field of swarm optimization.

Chapter 7

Bayesian Optimization Models for Particle Swarms

Published in Proceedings of GECCO 2005, Volume 1, pages 193–200

Abstract

We explore the use of information models as a guide for the development of single objective optimization algorithms, giving particular attention to the use of Bayesian models in a PSO context. The use of an explicit information model as the basis for particle motion provides tools for designing successful algorithms. One such algorithm is developed and shown empirically to be effective. Its relationship to other popular PSO algorithms is explored and arguments are presented that those algorithms may be developed from the same model, potentially providing new tools for their analysis and tuning.

7.1 Introduction

Particle Swarm Optimization (PSO) is a social or evolutionary optimization algorithm that was discovered during experiments with simulated bird flocking [Kennedy and Eberhart 1995]. The discovery was valuable, as it has proven to be a good approach to the optimization of a useful class of functions. It has the additional benefit that it is easy to implement and has relatively few tunable parameters.

As the reader is assumed to have some familiarity with PSO, the explanation that follows will be brief. Classical PSO begins by scattering particles in the function domain space, often by means of a uniform distribution. Each particle is a data structure that maintains its current position $\widehat{\mathbf{x}}$ and its current velocity $\widehat{\dot{\mathbf{x}}}$. Additionally, each particle remembers the most fit position it has obtained in the past, denoted $\mathbf{p}$ for "personal best". The most fit $\mathbf{p}$ among all particles is written $\mathbf{g}$ for "global best".

Each particle updates its location over time using

$$\widehat{\dot{\mathbf{x}}}_{t+1} = \widehat{\dot{\mathbf{x}}}_t + \phi_1 \, \mathrm{U}()(\mathbf{p} - \widehat{\mathbf{x}}_t) + \phi_2 \, \mathrm{U}()(\mathbf{g} - \widehat{\mathbf{x}}_t) \tag{7.1}$$

$$\widehat{\mathbf{x}}_{t+1} = \widehat{\mathbf{x}}_t + \widehat{\dot{\mathbf{x}}}_{t+1} \tag{7.2}$$

where usually $\phi_1 = \phi_2 = 2$, U() is drawn from a standard uniform distribution (either a scalar or a vector of random values to be applied to each element), and velocities are constrained to be smaller than some $V_{\max}$. Two simple and popular improvements to the technique are the use of a *constriction coefficient* χ [Clerc and Kennedy 2002] and an *inertia weight* ω [Shi and Eberhart 1998a], respectively:

$$\widehat{\dot{\mathbf{x}}}_{t+1} = \chi\Big(\widehat{\dot{\mathbf{x}}}_t + \phi_1 \, \mathrm{U}()(\mathbf{p} - \widehat{\mathbf{x}}_t) + \phi_2 \, \mathrm{U}()(\mathbf{g} - \widehat{\mathbf{x}}_t)\Big) \tag{7.3}$$

$$\widehat{\dot{\mathbf{x}}}_{t+1} = \omega\widehat{\dot{\mathbf{x}}}_t + \phi_1 \, \mathrm{U}()(\mathbf{p} - \widehat{\mathbf{x}}_t) + \phi_2 \, \mathrm{U}()(\mathbf{g} - \widehat{\mathbf{x}}_t) \ . \tag{7.4}$$

When using (7.3), definitions of ϕ_1 and ϕ_2 are different from above, and they are used to calculate χ.

Though improvements have been made on nearly all aspects of PSO, the basic structure of the motion equations has remained largely uncontested, limiting most motion improvements to the addition or tuning of equation coefficients. While some have deviated significantly from classical motion with good success [Clerc 2003; Kennedy 2003], few have attempted such a departure. Indeed, it seems that variants on classical motion are hard to beat, but in spite of much analysis of convergence and other motion characteristics [Clerc 1999; Clerc and Kennedy 2002; Ozcan and Mohan 1999; Eberhart and Shi 2000], as well as some valuable intuition [Kennedy and Eberhart 2001; Mendes et al. 2003], little is known as to why this is true.

In a way, the serendipitous origins of classical PSO have never been fully overcome, leaving room for a more principled and high-level perspective for PSO motion. This paper presents and motivates a model-oriented approach to particle motion algorithms, providing tools for the creation of such algorithms that also aid in their analysis. This approach makes explicit the information relationships and optimization assumptions that go into the design of a swarm optimization algorithm.

One class of optimization models based on Bayesian influence networks is presented first. An algorithm is then produced by solving one such model. This algorithm has some deficiencies, which are addressed by making better use of available information and applying approximations to its most complex features. The approximate algorithm is shown to perform well against some recent and successful PSO techniques. Finally, the ramifications of the new algorithm and the process by which it was created are explored.

7.2 Bayesian Optimization Models

Even though "No Free Lunch" (NFL) theorems dictate that no single algorithm can be used to efficiently optimize every class of functions [Wolpert and Macready 1997], any optimization problem can be modeled. The purpose of such a model is to sort out all of the information available during the optimization process and to make use of that information in a principled way. That it is always possible to apply a model neither ignores nor negates NFL, but rather indicates that the model must somehow explicitly specify the class of functions that are interesting.

Many such optimization models are possible, but this paper will define and restrict itself to a limited class of these models, hereafter referred to as Bayesian Optimization Models (BOMs). In a BOM, the optimization problem is framed as inference in a Dynamic Bayesian Network (DBN) where information relationships are characterized as conditional probability distributions [Russel and Norvig 2003]. While no assertion is made that DBNs are the only appropriate modeling tool for PSO, some class of models must be chosen, and DBNs have some particularly convenient properties.

The use of DBNs instead of a more general class of models represents an approximation that is open to debate. It is a fact, however, that an approximation of some kind must be made, since a fully expressive model of every detail of all possible information relationships would not be tractable for descriptive or computational purposes. Throughout this paper many choices will be made for the sake of approximation, and all of them will be explicit. In the interest of coherent

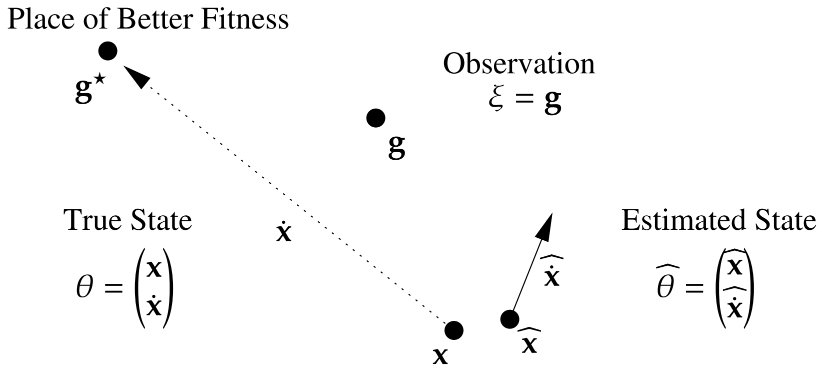

Place of Better Fitness
$\mathbf{g}^\star$
Observation
$\xi = \mathbf{g}$
$\mathbf{g}$
True State
$\theta = \begin{pmatrix}\mathbf{x}\\ \dot{\mathbf{x}}\end{pmatrix}$
$\dot{\mathbf{x}}$
$\widehat{\dot{\mathbf{x}}}$
Estimated State
$\widehat{\theta} = \begin{pmatrix}\widehat{\mathbf{x}}\\ \widehat{\dot{\mathbf{x}}}\end{pmatrix}$
$\mathbf{x}$
$\widehat{\mathbf{x}}$

Figure 7.1: Raw information available to swarm optimization

exposition, however, a detailed discussion of those choices will be deferred until Section 7.6.

DBNs are frequently used to characterize time-sensitive relationships between observable (ξ) and hidden (θ) variables or states. Usually the hidden variables represent desired information and are considered to cause or influence the state of the observable variables. For example, determining the location of an airplane given a blip on a radar screen fits this model: the true location of the plane is unknown or *hidden*, it causes an *observable* blip, and the data is time-sensitive. Provided that a useful *observation model* is available to characterize the noise and other behavior of the radar, the true state can be inferred with some accuracy from a series of observed blips.

In swarm optimization, each particle may observe $\mathbf{g}$ and $\mathbf{p}$. In the context of a BOM, these observations are influenced by some state θ encoding desirable but hidden information, in this case instructions as to what the particle should do to get to a place of even better fitness $\mathbf{g}^\star$.

Ignoring $\mathbf{p}$ for the time being, Figure 7.1 illustrates the use of hidden states and the raw information available to a swarm optimization algorithm. At each time step, a particle has a current estimate $\widehat{\theta}$ of its hidden state θ. Since the particle is trying to track the trajectory toward $\mathbf{g}^\star$, this estimate is encoded in the actual

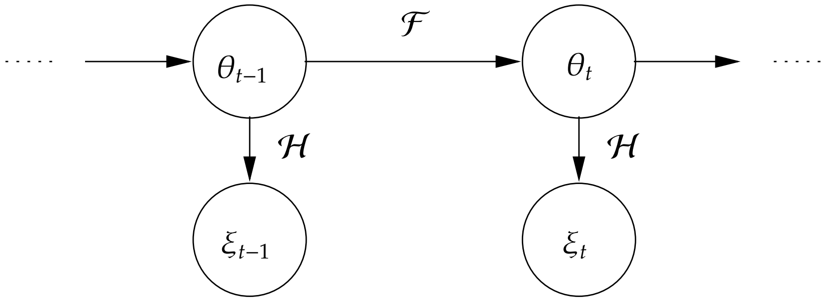


Figure 7.2: Hidden Markov Model

position and velocity of the particle and is gleaned from data available in the form of **g**. The hidden state θ represents where the particle ideally *should have been* ($\mathbf{x}$) and the direction in which it *should be going* ($\dot{\mathbf{x}}$) to get directly to a place of even better fitness $\mathbf{g}^\star$.

Just as a radar's observation model is required in order to estimate a plane's true location given a blip, the relationship between θ and ξ must be specified for a particle to effectively estimate a trajectory toward $\mathbf{g}^\star$. Figure 7.2 illustrates one of the simplest possible models for the various relationships in this system. In this case what is depicted is actually an instance of a Hidden Markov Model (HMM) where the hidden state θ influences the observable state ξ using a known *observation model* $\mathcal{H}$ and changes over time according to some *transition model* $\mathcal{F}$.

In the radar example, $\mathcal{F}$ might be a model of how a plane is expected to move. Large planes do not change speed or direction very quickly and therefore might use a constant velocity transition model, treating deviations from constant velocity as admissible noise. In the context of PSO, the transition model describes the way that **g** is expected to move over time. It also describes, however, the way that particles *prefer* to move, since they will make use of $\mathcal{F}$ as they attempt to track the trajectory of **g** over time. The $\mathcal{F}$ chosen will therefore depend on the best swarm behavior for the class of functions subject to optimization. In this paper, we will focus on a constant velocity $\mathcal{F}$ for the sake of simplicity and for historical reasons, though other relationships are certainly possible.

The model also indicates that $\mathbf{g}^\star$ influences the observed $\mathbf{g}$ since hidden state influences observations. This influence is inherently noisy because it is unreasonable to believe with absolute certainty that the observation of $\mathbf{g}$ precisely pinpoints $\mathbf{g}^\star$. Noise is thus used as a model of *subjective uncertainty* about the usefulness or accuracy of an observation. In a function with local smoothness, a Gaussian distribution is a good model of uncertainty. For example, a high-variance Gaussian with mean $\mathbf{g}$ indicates that $\mathbf{g}$ is a useful indicator of $\mathbf{g}^\star$, but that our *belief* or *confidence* in that result is not very strong. Similarly, a low variance would indicate greater confidence that $\mathbf{g}$ is a likely place to look for $\mathbf{g}^\star$.

Regardless of the particular distribution chosen to represent belief, the use of a BOM defines the goal of optimization as the inference of a belief distribution over θ. Ideally, that distribution will converge over time to a delta function (or in the Gaussian case, a distribution with variance approaching zero) centered at the global minimum, but the amount of information available does not often allow for so much precision or certainty. The goal becomes one of finding a distribution that gets as close as possible to the truth and that represents as much confidence as possible in that estimate. This clarifies the role of the network in Figure 7.2: if the influences shown can be characterized, noise and all, then standard approaches to solving HMMs may be used to estimate a distribution over θ (the best action a particle can take) at each time step.

Every choice made in the creation of this particular BOM, and even the use of a BOM in the first place, represents some assumptions about the class of functions to be optimized. Sometimes the connection is readily evident and other times it is more subtle. The choice of a constant velocity model, for example, indicates that velocity contains information. A consequence of this is that fitness and distance from the global minimum should be correlated; if velocity contains information then in some sense distance must as well. Also, a Gaussian noise model introduces the assumption of some local smoothness in the function, since the belief distribution assigns similar weight to nearby regions.

Whatever the specific details, an optimization model will encompass raw information and intuition as illustrated in Figure 7.1 as well as explicit information relationships as illustrated in Figure 7.2. Together, these give an algorithm designer an opportunity to be explicit about not only the available information and what it means, but about the relationships that exist within it. The choice of BOMs in particular allows designers of swarm optimization algorithms to leverage the considerable body of existing knowledge about DBNs and HMMs to generate particle motion.

It should be reiterated that although this paper focuses almost exclusively on HMMs as models of swarm optimization, this restriction is not a requirement. Much richer models, probabilistic or otherwise, may also be applied.

7.3 A BOM Motion Algorithm

Solution methodologies for HMMs are plentiful, but perhaps none is so easily applied as the Kalman Filter. The Kalman Filter is directly applicable to the solution of an HMM like that in Figure 7.2, imposing the additional constraints of linear relationships and additive Gaussian noise [Kalman 1960; Russel and Norvig 2003; Leondes 1970]. This restricted subclass of HMMs is commonly known as Linear Dynamic Systems (LDSs).

This suggests that the Kalman Filter might be applied to PSO as a means of moving particles around, an idea that was introduced earlier in the KSwarm algorithm [Monson and Seppi 2004]. Its original presentation was unmotivated, however, making it difficult to determine whether and in what situations it may be sensibly applied. The BOM developed thus far supplies the needed motivation, since the application of a Kalman Filter to the model produces an algorithm very much like KSwarm. To assist in further development of this idea, the basics of the original KSwarm will be presented briefly.

The purpose of the Kalman Filter is to estimate a mean $\bar{\theta}_t$ and covariance $\mathbf{\Sigma}_t$ for the hidden state of an HMM given a series of observations ξ_t, which in this case correspond to measurements of $\mathbf{g}$. The mean may be viewed as the estimate of the hidden state and the covariance as a measure of confidence in that estimate. This estimate can be obtained through recursive application of the following equations, which find themselves at the heart of KSwarm:

$$\begin{aligned}
\mathbf{K}_t &= (\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta)\mathbf{H}^\top\Big(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta)\mathbf{H}^\top + \mathbf{\Sigma}_\xi\Big)^{-1} && (7.5)\\
\bar{\theta}_t &= \mathbf{F}\bar{\theta}_{t-1} + \mathbf{K}_t(\xi_t - \mathbf{H}\mathbf{F}\bar{\theta}_{t-1}) && (7.6)\\
\mathbf{\Sigma}_t &= (\mathbf{I} - \mathbf{K}_t\mathbf{H})(\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta) \ . && (7.7)
\end{aligned}$$

A single sample from the distribution over a prediction provides the new state of a particle, comprising a position and velocity:

$$\widehat{\theta}_{t+1} = \begin{pmatrix} \widehat{\mathbf{x}}_{t+1} \\ \widehat{\dot{\mathbf{x}}}_{t+1} \end{pmatrix} \sim \text{Normal}(\mathbf{F}\bar{\theta}_t, \mathbf{\Sigma}_t) \ . \qquad (7.8)$$

The value of ξ is obtained from observations, but several other values must be specified before the algorithm can proceed. For example, a transition matrix $\mathbf{F}$ and observation matrix $\mathbf{H}$ must be specified, as well as a prior mean $\bar{\theta}_0$. A constant velocity transition model makes the assumption that the trajectory traced by successive values of $\mathbf{g}$ will tend to take a straight line path toward the global minimum, and a skew-free observation model is used because it is unreasonable to assume that $\mathbf{g}$ is *not* a good estimate of $\mathbf{g}^\star$ in the absence of more information.

Additonally, the priors are taken directly from the initial location and velocity of the particle. These specifications are given together here:

$$\mathbf{F} = \begin{pmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \tag{7.9}$$

$$\mathbf{H} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \tag{7.10}$$

$$\bar{\theta}_0 = \begin{pmatrix} \widehat{\mathbf{x}_0} \\ \widehat{\dot{\mathbf{x}}_0} \end{pmatrix} \quad . \tag{7.11}$$

Each of these values carries with it a corresponding covariance matrix, denoted $\mathbf{\Sigma}_\theta$, $\mathbf{\Sigma}_\xi$, and $\mathbf{\Sigma}_0$, respectively. Given a vector $\mathbf{w}$ of side lengths of the "feasible rectangle" (usually provided with the target function) and a small constant ϵ (≈ 0.001), the original KSwarm attempts to simplify the creation of useful covariances by defining diagonal matrices for these values, thus:

$$\mathbf{\Sigma}_\theta = \epsilon \operatorname{diag} \begin{pmatrix} \mathbf{w} \\ \mathbf{w} \end{pmatrix} \tag{7.12}$$

$$\mathbf{\Sigma}_\xi = \epsilon \operatorname{diag}(\mathbf{w}) \tag{7.13}$$

$$\mathbf{\Sigma}_0 = \epsilon \operatorname{diag} \begin{pmatrix} \mathbf{w} \\ \mathbf{w} \end{pmatrix} \quad . \tag{7.14}$$

Though KSwarm's published implementation outperforms one version of constricted PSO on a number of common benchmark functions, it has not been shown to be competitive with more recent or well-tuned PSO algorithms.

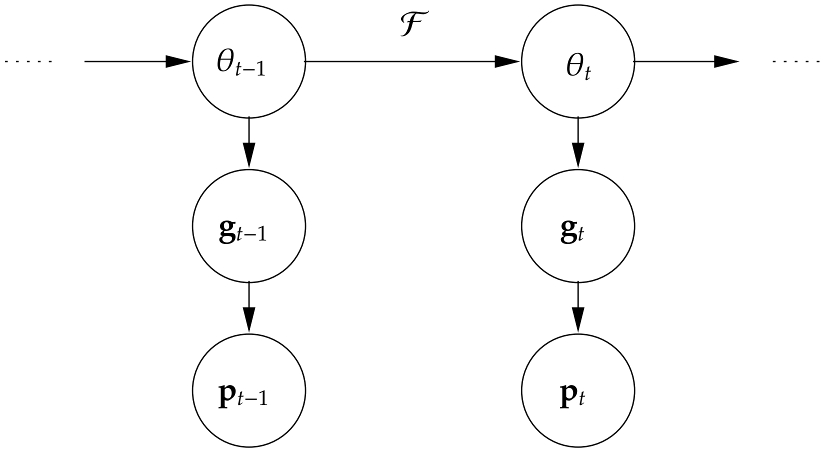


Figure 7.3: Belief network with **p** included

7.4 Taking the Next Step

KSwarm was produced from a BOM, making available a large number of existing tools for its analysis. It is, however, not perfect, and several improvements immediately suggest themselves. First, KSwarm ignores the existence of **p**, which is commonly considered to be an important piece of information in PSO [Kennedy and Eberhart 2001; Kennedy 2003]. Second, the computational complexity of KSwarm is $O(D^3)$ while other popular algorithms are $O(D)$. Unfortunately, its performance does not warrant this increase in complexity. This section addresses both of these problems.

7.4.1 Incorporating Personal Best

There are several ways to introduce **p** into the HMM previously outlined. Perhaps the simplest is to combine **p** and **g** into a single observation before applying the Kalman Filter. Thus, $\xi_t = C(\mathbf{g}, \mathbf{p})$, where C is some function that combines the two pieces of information in a useful way.

If **p** is considered to be dependent on **g**, the new network is represented in Figure 7.3. Combining **g** and **p** into a single observation is equivalent to letting

$\mathbf{p}$ temper the perception of $\mathbf{g}$, which is accomplished by finding the posterior distribution $G'(\mathbf{g}|\mathbf{p})$ by application of Bayes' Law:

$$C(\mathbf{g}, \mathbf{p}) \sim G'(\mathbf{g}|\mathbf{p}) = \frac{P'(\mathbf{p}|\mathbf{g})G(\mathbf{g})}{P(\mathbf{p})} \tag{7.15}$$

where G and G' are the prior and posterior distributions over $\mathbf{g}$, and P and P' are the marginal and conditional distributions over $\mathbf{p}$. When using multivariate Gaussian distributions, $G'(\mathbf{g}|\mathbf{p})$ is parameterized by mean $\bar{\mathbf{b}}$ and covariance $\mathbf{\Sigma_b}$, the values of which are well known:

$$\mathbf{W} = \mathbf{\Sigma_p}(\mathbf{\Sigma_p} + \mathbf{\Sigma_g})^{-1} \tag{7.16}$$

$$\bar{\mathbf{b}} = (\mathbf{I} - \mathbf{W})\mathbf{p} + \mathbf{W}\mathbf{g} \tag{7.17}$$

$$\mathbf{\Sigma_b} = (\mathbf{I} - \mathbf{W})\mathbf{\Sigma_p} \tag{7.18}$$

where $\mathbf{\Sigma_p}$ and $\mathbf{\Sigma_g}$ represent uncertainty about the utility of $\mathbf{p}$ and $\mathbf{g}$ as estimates of $\mathbf{g}^\star$, respectively.

Substituting $\bar{\mathbf{b}}$ and $\mathbf{\Sigma_b}$ for ξ_t and $\mathbf{\Sigma}_\xi$ in (7.5), (7.6), and (7.7) yields these equations for the "$\mathbf{p}$-augmented KSwarm":

$$\mathbf{K}_t = (\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta)\mathbf{H}^\top\Big(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta)\mathbf{H}^\top + \mathbf{\Sigma_b}\Big)^{-1} \tag{7.19}$$

$$\bar{\theta}_t = \mathbf{F}\bar{\theta}_{t-1} + \mathbf{K}_t\Big((\mathbf{I} - \mathbf{W})\mathbf{p} + \mathbf{W}\mathbf{g} - \mathbf{H}\mathbf{F}\bar{\theta}_{t-1}\Big) \tag{7.20}$$

$$\mathbf{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})(\mathbf{F}\mathbf{\Sigma}_{t-1}\mathbf{F}^\top + \mathbf{\Sigma}_\theta) \ . \tag{7.21}$$

The equations do not change the order of polynomial complexity of the original KSwarm, but they do require the specification of yet another covariance matrix.

7.4.2 The Necessity of Approximations

The use of the Kalman Filter incurs some costs. It was mentioned briefly that although it tests well against a version of constricted PSO, it does not test as well against more recent improvements. This may be due to the difficulty inherent in tuning an algorithm whose parameters are all large matrices. In fact, even though some intuition may be applied to the specification of covariance matrices, the dimensionality of the parameter space far exceeds the dimensionality of the function to be optimized!

In addition, the increased computational complexity must be addressed. Fortunately, it is possible to address both issues simultaneously by crafting an approximation to the **p**-augmented KSwarm algorithm. Of particular interest is (7.20) which can be rewritten as follows:

$$\bar{\theta}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{F}\bar{\theta}_{t-1} + \mathbf{K}_t(\mathbf{I} - \mathbf{W})\mathbf{p} + \mathbf{K}_t\mathbf{W}\mathbf{g} \ . \tag{7.22}$$

To simplify things further, it may be assumed that $\mathbf{H} = \mathbf{I}$ and that observation vectors are appropriately augmented with a velocity estimate:

$$\left(\bar{\mathbf{x}}_t\bar{\dot{\mathbf{x}}}_t\right) = (\mathbf{I} - \mathbf{K}_t)\mathbf{F}\left(\bar{\mathbf{x}}_{t-1}\bar{\dot{\mathbf{x}}}_{t-1}\right) + \mathbf{K}_t(\mathbf{I} - \mathbf{W})\left(\mathbf{p}\mathbf{p} - \widehat{\mathbf{x}}_{t-1}\right) + \mathbf{K}_t\mathbf{W}\left(\mathbf{g}\mathbf{g} - \widehat{\mathbf{x}}_{t-1}\right) \ . \tag{7.23}$$

Because a new position may be trivially computed given the previous position and a new velocity, (7.23) may be further simplified by dropping all portions of the equation that directly calculate a position. Recalling from (7.9) that **F** preserves velocity allows it to be dropped entirely from the calculation of the mean filtered velocity $\bar{\dot{\mathbf{x}}}_t$:

$$\bar{\dot{\mathbf{x}}}_t = (\mathbf{I} - \mathbf{K}_{t,v})\bar{\dot{\mathbf{x}}}_{t-1} + \mathbf{K}_{t,v}(\mathbf{I} - \mathbf{W}_v)(\mathbf{p} - \widehat{\mathbf{x}}_{t-1}) + \mathbf{K}_{t,v}\mathbf{W}_v(\mathbf{g} - \widehat{\mathbf{x}}_{t-1}) \ . \tag{7.24}$$

Thus, $\bar{\mathbf{x}}_t$ looks like a convex combination of $\bar{\mathbf{x}}_{t-1}$, $\mathbf{g}$, and $\mathbf{p}$. Even more simplification is possible if the gains $\mathbf{K}$ and $\mathbf{W}$ are assumed to be the constant scalars a and b instead of dynamic matrices:

$$\bar{\mathbf{x}}_t = (1-a)\widehat{\mathbf{x}}_{t-1} + ab(\mathbf{p} - \widehat{\mathbf{x}}_{t-1}) + a(1-b)(\mathbf{g} - \widehat{\mathbf{x}}_{t-1}) \tag{7.25}$$

where $\bar{\mathbf{x}}_{t-1}$ is approximated by $\widehat{\mathbf{x}}_{t-1}$. This bears some resemblance to (7.4), the equation for inertia-weighted PSO.

The variance vector is all that remains to be determined to make the algorithm concrete. A fairly common trick when adding noise at the end of a PSO calculation is to let the variance be a scalar based on the magnitude of the mean velocity [Clerc 2003; Kennedy 2003]. Without further motivation, a similar trick is applied here:

$$\widehat{\mathbf{x}}_t \sim \mathrm{Normal}\left(\bar{\mathbf{x}}_t, \psi\frac{\|\bar{\mathbf{x}}_t\|^2}{D}\mathbf{I}\right) \tag{7.26}$$

where ψ scales the calculated variance by some fixed amount, usually a small number like 0.05. The dimensional scaling attempts to counter an explosion of the Gaussian support volume as dimensionality increases.

Given the mean as defined in (7.25) and variance from (7.26), a new velocity may be created by sampling once from a Gaussian distribution. This approach is the basis of the "**p** Approximate Kalman Swarm" (PAKS) algorithm, which restores $O(D)$ complexity and substantially reduces the size of the parameter space.

The performance of this algorithm is compared with some popular PSO enhancements on several benchmark functions in Figure 7.4. The benchmarks are defined in Table 7.1. The results were obtained for minimization in 30 dimensions, $\alpha = 0.45$, $\beta = 0.5$, $\psi = 0.05$, and a non-reflexive "star" topology (fully connected but without self links) with 20 particles where applicable. TRIBES began with a single

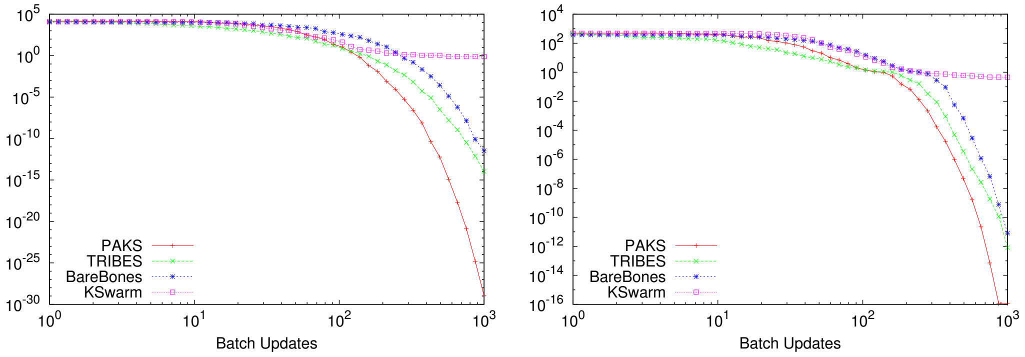
Table 7.1: Common benchmark functions

$$\begin{aligned}
\text{Sphere}(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 \\
\text{Griewank}(\mathbf{x}) &= \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \\
\text{Rosenbrock}(\mathbf{x}) &= \sum_{i=1}^{D-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \\
\text{DeJongF4}(\mathbf{x}) &= \sum_{i=1}^{D} i x_i^4 \\
\text{Rastrigin}(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 + 10 - 10\cos(2\pi x_i)
\end{aligned}$$

particle. In the case of TRIBES, the x-axis represents the best value after every 20 function evaluations to make the results directly comparable. As can be seen, PAKS either outperforms or remains competitive with TRIBES and BareBones. Although it is only an approximation, it always outperforms KSwarm, presumably because it is much easier to tune.

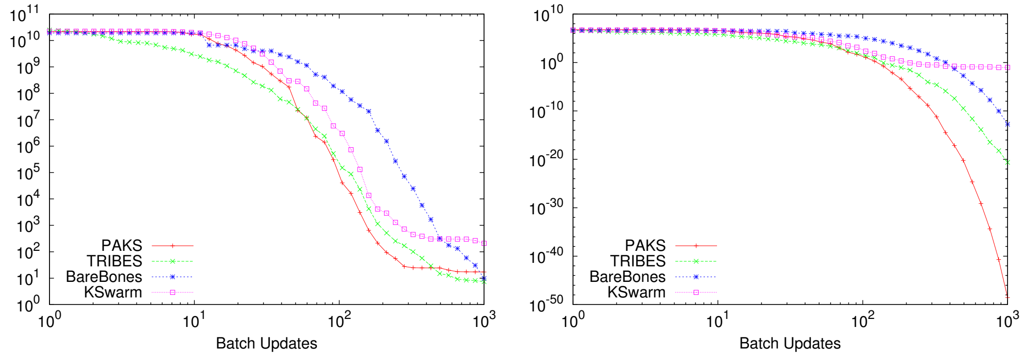
Although PAKS does require more tuning than TRIBES or BareBones, even the most naive parameter settings produced good behavior. Setting $\beta = 0.5$ naively assumes that **g** and **p** are equally reliable sources of information. This was also tried as the value of α, but velocity explosion required it to be lowered slightly. The only parameter that really required any significant attention was ψ, which was reasonably robust once the right range was found.

More significant than the naive and simplistic nature of the tuning is the fact that it could easily have been more principled. The use of a BOM as the basis for PAKS provides a clear and explicit path from a statistical model to a concrete algorithm. That it has roots in a model allows better tuning to occur before the



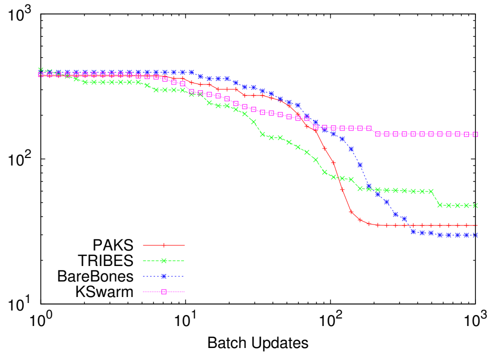
(a) Sphere

(b) Griewank



(c) Rosenbrock

(d) DeJongF4



(e) Rastrigin

Figure 7.4: Average fitness of PAKS compared with KSwarm, TRIBES, and Bare-Bones

algorithm has ever been executed. The coefficients, for example, are rooted in subjective variance, which was never overtly used. It is conceivable that some experiments could shed light on appropriate variances and thus on appropriate choices of coefficients, something that would not be possible without the use of the model as a starting point. Even without such analysis, it is significant that a simple algorithm using a topology not noted for its exploration capabilities can perform so favorably.

7.5 Applications of a BOM

BOMs are useful tools for specifying motion algorithms, but they represent just one possible class of models. Perhaps even more interesting than the presented BOM is the accompanying process that was used to generate useful particle swarm motion; the model is the starting point, the solution methodology creates a real algorithm, and the final approximation makes that algorithm tractable. Together these ideas represent a unified framework for function optimization that provides insights into how to tune the new algorithm and why it behaves the way that it does. All of the behavior of the algorithm may be traced back to one or more of the choices made during this process, all of which are explicit, allowing any desired change to be affected by revisiting those choices.

The full impact of this idea becomes evident when working backwards from existing motion methodologies to find appropriate models. In many cases, the very same BOM may be obtained in this manner. Consider, for example, the similarities between PAKS and other popular techniques for computing a new particle velocity:

Constricted [Clerc and Kennedy 2002]:

$$\chi\Big(\widehat{\dot{\mathbf{x}}}_t + \phi_1\,\mathrm{U}()(\mathbf{p} - \widehat{\mathbf{x}}_t) + \phi_2\,\mathrm{U}()(\mathbf{g} - \widehat{\mathbf{x}}_t)\Big)$$

Inertia-weighted [Shi and Eberhart 1998a]:

$$\omega\widehat{\dot{\mathbf{x}}}_t + \phi_1\,\mathrm{U}()(\mathbf{p} - \widehat{\mathbf{x}}_t) + \phi_2\,\mathrm{U}()(\mathbf{g} - \widehat{\mathbf{x}}_t)$$

Noisy Classical [Clerc 2003]:

$$\chi\Big(\widehat{\dot{\mathbf{x}}}_t + \mathrm{G}(\mathbf{p} - \widehat{\mathbf{x}}_t, \mathbf{I}\frac{||\mathbf{p} - \widehat{\mathbf{x}}_t||^2}{4}) + \mathrm{G}(\mathbf{g} - \widehat{\mathbf{x}}_t, \mathbf{I}\frac{||\mathbf{g} - \widehat{\mathbf{x}}_t||^2}{4}))\Big)$$

PAKS (mean velocity):

$$(1 - a)\widehat{\dot{\mathbf{x}}}_t + ab(\mathbf{p} - \widehat{\mathbf{x}}_t) + a(1 - b)(\mathbf{g} - \widehat{\mathbf{x}}_t)$$

($\mathrm{G}(\cdot, \cdot)$ produces a sample from a Gaussian with the supplied mean vector and covariance matrix).

The similarity of the algorithms is striking, as each performs a noisy and linear combination of the same information. In the case of PAKS, it is clear why this is the case: Bayesian reasoning, linearity, and Gaussian noise were elective constraints during the design process. That it is similar to other popular PSO algorithms implies similarity in their underlying model and solution methodology. Consequently, all of these algorithms are describable as approximations of $\mathbf{p}$-augmented KSwarm. The differences are limited to tuning or noise insertion and are generally superficial.

Knowing the model that can produce these algorithms makes available a wealth of information about them that was not as easily accessible before. Consider, for example, the problem of a varying inertia coefficient (ω) for inertia-weighted PSO. If we view inertia-weighted PSO as an approximation to KSwarm, some interesting things can be said about ω, which corresponds directly to the $\mathbf{I} - \mathbf{K}$ term in the Kalman Filter equations. Because $\mathbf{\Sigma}_t$ converges over time to a fixed point, a large $\mathbf{\Sigma}_0$, corresponding to low confidence in the initial estimate, will cause $\mathbf{\Sigma}_t$ to decrease over time. This decreasing variance corresponds to an *asymptotically increasing* inertia weight, unlike the *linearly decreasing* values so commonly used. While this violates some popular intuition, the superiority of an increasing inertia weight has already been empirically observed [Zheng et al. 2003].

That the noise models are not precisely equivalent is no barrier to forging a connection between existing algorithms and PAKS, especially in consideration of the Central Limit Theorem. Some of them use uniform distributions on the coefficients themselves, but a sum of uniform random variables is Gaussian in the limit, a notion verified empirically in other works [Kennedy 2003; Clerc 2003]. Additionally, a sum of Gaussian variables produces a new Gaussian variable, so taking a single sample at the end is not fundamentally different from taking samples on each coefficient. Perhaps the most compelling reason to suggest that all of these algorithms use a Gaussian noise model is the fact that they all combine information linearly, a consequence of the closure of Gaussians under Bayes' Law. Any other noise model would produce a more complex combination of information.

The use of BOMs also applies to motion that does not share as much in common with PAKS as the above algorithms. For example, BareBones, which performs a simple position computation based on $\mathbf{p}$ and $\mathbf{g}$ [Kennedy 2003], may be produced from the model shown in Figure 7.3 by removing the time-sensitive links and dropping velocity from the approximation rather than position. Similar modifications are possible for other algorithms not discussed here.

7.6 Modeling and Approximations

While progressing from an optimization model to a concrete PSO algorithm, several modeling decisions and approximations were made along the way. Every one of these decisions was explicit, which is one of the more important benefits of using the design framework outlined here: explicit decisions are easy to analyze and adjust. In spite of the clarity resulting from this process, however, some of the specific decisions and approximations made here have subtle consequences and merit further discussion.

7.6.1 Bayesian Modeling

Although DBNs do not represent the only class of models that can be used to describe information relationships, it has been made clear that simplicity and tractability were the reasons behind using that class of models. What is perhaps not as clear is the intuitive meaning of the hidden processes in Figures 7.1 and 7.2 and the approximations made in ignoring some of the possible information relationships.

Because PSO is a known and concrete algorithm, the only truly hidden process is the target function itself. The HMM presented here approximates all of the hidden characteristics of the target function as a sequence of trajectories. Intuitively, this means that the features of the target function deemed most important by the model are described by paths of improving fitness from each point in space, making a particle's goal one of finding the next step along such a path from its current location. Inherent in this is the assumption that greedy improvements to $\mathbf{g}$ form a noisy trajectory that will eventually lead to the global optimum. This assumption provides a partial specification of the class of functions for which this PSO algorithm is expected to be effective: it will work well on functions that can be described as unimodal with noise, an idea born out in the results for Sphere, DeJongF4, and Griewank.

In addition to approximating a hidden process, the proposed HMM ignores a number of possible information relationships. While adding more information links is certainly possible, limiting them to a select few is common when dealing with Bayesian networks since specifying all possible relationships is not often tractable. The underlying assumption is that noise is an adequate substitute for some of the less consequential relationships.

7.6.2 Adding New Information

The question naturally arises as to whether the approach to adding new information detailed in Section 7.4.1 is truly representative of what the information means. Is, for example, $\mathbf{p}$ really dependent upon $\mathbf{g}$? One could certainly argue that it is not, since no such dependency is built into PSO or into optimization in general. In fact, it is much easier to argue that such a relationship would work the other way, since $\mathbf{g}$ is actually the $\mathbf{p}$ of some particle.

This is readily addressed by looking at the model as a description of observed behavior rather than of a known process. While $\mathbf{g}$ does not *cause* the observation of $\mathbf{p}$, it can be *observed* that $\mathbf{p}$ tends to be close to $\mathbf{g}$ during a run of PSO.

The dependency of $\mathbf{p}$ on $\mathbf{g}$ can also be viewed as a *specification* of behavior. This observation of correlation or clustering among $\mathbf{p}$ and $\mathbf{g}$ is, in a sense, a self-fulfilling prophecy: the model says that clustering occurs, the same model drives the motion of particles, and therefore clustering is observed.

This brings up an interesting side point. Many choices were made in this paper not only because they were simple and convenient, but because they led to an algorithm that was similar to existing PSO algorithms. This allowed the framework to be developed in a familiar context and provided a means of rethinking popular motion algorithms in terms of a useful model. Remarkably little foresight was required to do this since some of the approximations made were rather obvious and naive. For these reasons, the seemingly counterintuitive dependency of $\mathbf{p}$ on $\mathbf{g}$ is actually not at all unreasonable; it is descriptive, prescriptive, and effectively models the behavior of existing algorithms.

7.6.3 Other Approximations

Other notable approximations in Section 7.4.2 include dropping position instead of velocity and ignoring the Kalman variance calculation in (7.21).

Position was dropped instead of velocity to better facilitate the connection between PAKS and other algorithms, especially inertia-weighted PSO. It would definitely be possible to drop velocity instead, producing something more like BareBones, as previously discussed.

As for PAKS variance, the corresponding Kalman equations were ignored for simplicity and space. This amounted to replacing useful information with a trick adapted from the literature, highlighting the fact that significant approximations are often made while building a PSO algorithm. In the case of PAKS, it was clear that the correct variance calculation was ignored, but it was only obvious because the underlying model was already known. Many PSO algorithms, on the other hand, commonly make equally sweeping assumptions without any context for their objective evaluation. Indeed, it is difficult if not impossible to provide such context without the use of a model.

7.7 Conclusions and Future Work

The single-objective optimization problem is one of using available information to find the global minimum. In order to do this, it is useful to specify in explicit terms what that information is and the relationships between various pieces of the available information. Bayesian Optimization Models are a class of models that make this specification systematic and principled, simultaneously lending valuable intuition to the process.

That this approach can serve as the foundation of a number of PSO motion algorithms, including those developed here, makes it useful as a tool for high level analysis of both new and existing PSO algorithms. It would be interesting to apply in greater detail the model and approximation techniques presented here to existing PSO algorithms. This could provide more insights into their differences in behavior on various functions.

Other DBN models may be used besides the HMM suggested here, and other solution methodologies, such as particle filters, may be applied. More information than **g** and **p** could also be made available to any chosen model, and methods of combining information should be explored more fully. Additionally, the approximation process outlined for PAKS could be changed to make better use of the model to make coefficient tuning more principled.

In this work, the application of a BOM has only affected each particle individually, but it is possible to create a richer model that includes the notion of sociometry and the information flow between particles in the swarm. Though complex, the study of such influences is at least possible using a model, and its use may provide new topological insights.

The algorithm design framework developed around BOMs actually exists independently of them, suggesting that any model, Bayesian or otherwise, may be used in the framework. The only requirement is that such a model make information relationships explicit and provide a means for inferring desired information from available information. The exploration of alternative models would be an interesting pursuit.

The introduction of a BOM and its success in creating a competitive PSO algorithm highlights the utility of the associated algorithm design framework. The framework is not only valuable as a tool for the synthesis of PSO algorithms, but also for their analysis. This work has presented the framework and model-based approach as a way of *thinking* about optimization and this perspective suggests new ways of approaching the problem.

Part III

Bayesian Frameworks for Utility-Based Optimization

The core contribution of this work is contained in Part III. While previous chapters focused on improving PSO and modeling optimization within that context, the chapters comprising Part III represent a departure from the consideration of any one evolutionary optimization approach. The lessons learned in previous chapters are applied: NFL and the associated function class are important enough to receive explicit attention in any optimization process, information models are useful tools in algorithm design, and the true goals of optimization must be directly considered to produce an algorithm that makes rational use of information.

The paper in Chapter 8 introduces a statistical perspective on optimization by modeling the function sampling process and its inherent goals. The result is a static Bayesian network in which inference can be performed to provide information about the location of the global optimum. Extending this network to the dynamic case seems trivial on the surface, but some of the new dynamic connections are in fact difficult to motivate.

This difficulty is addressed by the paper in Chapter 9. The problem with the dynamic relationships in the previous paper is addressed by incorporating explicit definitions of utility and cost into the optimization model. This critical extension to the idea finally allows the real goal of optimization to be expressed: obtaining information about the global optimum through careful sampling of the target function. The resulting algorithm is simple, general, and powerful: given specifications of the desired function class, output utility, and sample cost, it rationally dictates where the function should be sampled so that maximal information is obtained and properly utilized. Furthermore, the algorithm knows when exploration should cease and exploitation begin, a fundamental issue in any optimization setting.

Following these papers is a chapter that concludes the work, summarizing its key contributions and suggesting interesting avenues for future research.

Chapter 8

The Evolutionary Optimization DBN

In Review for the Evolutionary Computation Journal

Abstract

We present a statistical model of empirical optimization that admits the creation of algorithms with explicit and intuitively defined desiderata. Because No Free Lunch theorems dictate that no optimization algorithm can be considered more efficient than any other when considering all possible functions, the desired function class plays a prominent role in the model. In particular, this provides a tractable way to answer the traditionally difficult question of what algorithm is best matched to a particular class of functions. Among the benefits of the model are the ability to specify the function class in a straightforward manner, a natural way to specify noisy or dynamic functions, and a new source of insight into No Free Lunch theorems for optimization.

8.1 Introduction

Empirical function optimization, as addressed in this work, is the process of sampling a continuous cost function at discrete locations to search for a vector that produces the global minimum. In recent years, evolutionary algorithms have become a popular and powerful tool in both discrete and continuous settings, including such approaches as Genetic Algorithms (GAs) [Holland 1975; Goldberg 1989], Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995], Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano 2001], and others. This work will focus on and assume the continuous case, although application to discrete problems is supported.

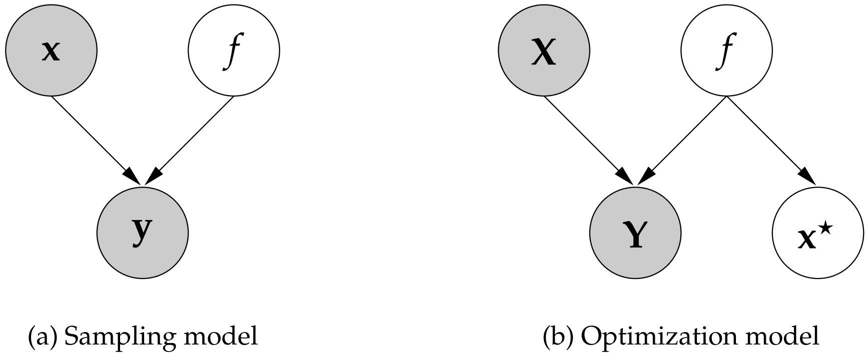
While evolutionary algorithms generally attempt to be good black-box function optimizers, No Free Lunch (NFL) theorems indicate that no single algorithm can efficiently optimize the set of all possible functions [Macready and Wolpert 1996; Wolpert and Macready 1997]. While these theorems were developed and proven for discrete problems, it is widely believed that they also apply to continuous applications. This poses an interesting problem for researchers and algorithm designers: any given approach to optimization is known *a priori* to be ineffective on a large class of functions. This lends significance to an interesting and often elusive question: on which class of functions will a given algorithm perform well? Ideally, it will work well on a *useful* class of functions, generally meaning that it is readily and efficiently applied to some set of important real-world problems; any problem not in the class is of no interest.

Finding such an algorithm–class pairing is usually a difficult task, leaving researchers few options but to gather empirical evidence to support or refute a pairing hypothesis. Such evidence is generally collected by applying the algorithm to a set of standard benchmarks, commonly chosen so that various high-level function characteristics are represented in the tests: continuity, deceptiveness, smoothness, symmetry, etc. This approach, while properly recognizing the need to establish algorithm–class pairings, does not necessarily achieve its goal [Whitley and Watson 2006].

At least partly to blame for the inherent difficulty of establishing good pairings is the fact that evolutionary algorithms are traditionally motivated by intuition or biological processes; while often useful, such motivation differs enough from the question of algorithm–class pairing that it sheds little light upon it: the algorithm's design desiderata, e.g. "a good hill climber" or "able to jump over local minima", do not directly reflect its corresponding performance desiderata, e.g., "an efficient optimizer of noisy bowl-shaped functions" or more generally "tuned to function class $\mathcal{F}$". Because of No Free Lunch, any algorithm designed with the goal of outperforming random search must encode the function class for which it is expected to do so; unfortunately, that specification is often hidden within the machinery of the algorithm; the function class is precisely but blindly designed into the method.

While recent approaches like EDAs suffer less from this disconnect than more traditional evolutionary algorithms [Syswerda 1993; Baluja 1994; Baluja and Caruana 1995; de Bonet et al. 1997; Pelikan et al. 1999; Pelikan and Goldberg 2000a,b; Pelikan et al. 2002], the distinction between design and performance desiderata remains. EDAs, which are motivated by assumptions of statistical closeness of good locations, continue to leave the true function class definition implied.

We propose an evolutionary optimization algorithm that, like EDAs, gathers and exploits statistical information obtained during the sampling process. The algorithm is based on statistical inference in a Dynamic Bayesian Network (DBN), and is called the Evolutionary Optimization DBN (EO-DBN). While this model uses population information to generate distributions from which new populations



(a) Sampling model (b) Optimization model

Figure 8.1: Basic sampling and optimization models

are sampled, it differs from traditional EDAs in that it both admits and requires an explicit and direct specification of the function class for its operation, thus elegantly unifying design and performance desiderata. In fact, the specification of function class represents nearly all of the prior information needed for the algorithm's operation.

We begin by describing simple static Bayesian networks that model function sampling and optimization, with more particular attention given to the latter. Several examples are then given which illustrate the process of inference and the significance of various aspects of the model. Finally, the full EO-DBN is created by introducing evolution into the optimization network, and its behavior is demonstrated and discussed.

8.2 Bayesian Models of Sampling and Optimization

The basic goal of using a Bayesian network to describe and facilitate optimization is the unification of design and performance desiderata in the resulting algorithm. More particularly, it is to give the desired efficiently-optimizable function class a prominent role in the process of algorithm design, not merely in post-design performance analysis.

To accomplish this goal, one must first consider the process at the heart of empirical optimization: function sampling. This process consists of three essential components:

- f: the function itself,
- $\mathbf{x}$: a location to be sampled, and
- $\mathbf{y}$: all information obtained at that location.

The relationship between these elements is depicted in Figure 8.1(a), which shows that f and $\mathbf{x}$ influence $\mathbf{y}$. When obtaining deterministic value samples, the nature of that influence is unambiguously defined as $\mathbf{y} = f(\mathbf{x})$, but the network also allows for more general relationships. A great deal of flexibility is also allowed in the contents of $\mathbf{y}$ (e.g., a value, a gradient, or both), but in this work it will generally be assumed that $\mathbf{y}$ is a singleton consisting only of the value at $\mathbf{x}$.

The network displays the function as an unshaded node, indicating that it is a variable whose value is hidden. Even though it must be available for sampling, it is considered "hidden" because a complete description of the *characteristics of interest* (such as the location of its global minimum) is not available. This is even true in many cases where an analytical form of the function is known; if the characteristics of interest are hidden, so is the function in the model.

When supplied with definitions of the various information relationships, Bayesian inference may be applied to the network to calculate distributions over any of its unknowns. It is possible, for example, to use the model to compute $\rho(\mathbf{y} \,|\, f, \mathbf{x})$, a distribution describing the likelihood of various outputs given a function and a point at which it should be sampled. More interesting, however, is the ability to infer $\rho(f \,|\, \mathbf{x}, \mathbf{y})$, a distribution over the functions that are consistent with an observed sample. This latter distribution encapsulates the goal of function approximation, e.g., learning a distribution over the weights of an artificial neural network [De Freitas et al. 2000].

Function approximation is interesting, but optimization does not require that the complete function be learned, limiting its goals instead to discovery of the global minimum located at $\mathbf{x}^\star$. The expanded network shown in Figure 8.1(b) is therefore more suitable for this purpose, including the goal of optimization directly into the network and clarifying its relationship to the other elements there: $\mathbf{x}^\star$ is defined or influenced by f. As long as a distribution describing this relationship is known, the network may be queried to determine $\rho(\mathbf{x}^\star \,|\, \mathbf{x}, \mathbf{y})$, which is the essence of empirical optimization: obtaining information about $\mathbf{x}^\star$ given one or more discrete samples.

While such a relationship between the true function and its global minimum is not generally known (else why optimize?), the relationship certainly exists and is therefore an essential part of the optimization model. Exactly how that relationship may be specified and used in practice will be discussed in detail later.

Finally, one more extension is present in Figure 8.1(b). Instead of merely representing the sampling of a single value $\mathbf{y}$ at a single location $\mathbf{x}$, the network describes the process of sampling multiple locations simultaneously, replacing $\mathbf{x}$ and $\mathbf{y}$ with the matrices $\mathbf{X}$ and $\mathbf{Y}$, respectively. This model will be referred to as the Bayesian Optimization Network (BON) for the remainder of this work.

8.3 Inference in the Bayesian Optimization Network

Optimization in the BON is achieved by inferring $\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y})$, a distribution over likely locations for the global minimum conditioned on information obtained from sampling. Inference of $\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y})$ in the Bayesian optimization model is accomplished using the Chain Rule and Bayes' Law [Russel and Norvig 2003]:

$$\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y}) = \int \rho(\mathbf{x}^\star \,|\, f)\rho(f \,|\, \mathbf{X}, \mathbf{Y})\, df \propto \int \rho(\mathbf{x}^\star \,|\, f)\rho(\mathbf{Y} \,|\, f, \mathbf{X})\rho(f)\, df \ . \tag{8.1}$$

This definition makes it clear that before inference can proceed, the *function prior* $\rho(f)$, *optimization distribution* $\rho(\mathbf{x}^\star \,|\, f)$, and *sampling distribution* $\rho(\mathbf{Y} \,|\, f, \mathbf{X})$ must be specified. Insofar as these specifications are representative of the intended function class and useful samples $\mathbf{X}$ and $\mathbf{Y}$ are obtained, $\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y})$ will be a good indicator for $\mathbf{x}^\star$, and sampling from it should produce fit locations, suggesting a way in which new populations might be generated.

Optimization researchers familiar with EDAs will note their similarity to this approach: a distribution over points likely to contain the global minimum is sampled to produce each new population. They will also, however, note the striking differences in the way that this distribution is obtained: an EDA creates its distribution from select members of a population by applying a predetermined distribution representation and acquisition methodology, while the Bayesian optimization model makes use of a function class definition as encoded in several distributions. EDAs rely on an *implicit* choice of function class, encoded in the distribution representation and the algorithm used to obtain it; and the Bayesian optimization model relies on an *explicit* choice of function class, encoded in the various distributions that describe the network. This distinction will be justified and discussed in greater detail throughout this work.

8.3.1 Practical Inference

It is not immediately clear how to specify a parametric prior over the space of all continuous functions that is both simple and useful, ruling out many analytical methods of inference, e.g., methods requiring Gaussian distributions. Numerical inference methods such Markov Chain Monte Carlo simulation or Particle Filtering (or Likelihood Weighting in static networks, though the distinction is not critical here) must be used instead [Russel and Norvig 2003]. Though based on discrete samples, particle filters are simple to implement and describe and are reasonably tractable with today's computing capabilities, making them the method of choice for this work's pedagogical purposes.

Particle filters sample "particles" from all prior distributions, then push the particles forward through related conditional distributions until encountering an observed variable. At that point, the likelihood of the observation is calculated with respect to each particle, providing a discrete estimate of the posterior distribution. This process consists roughly of the following steps, described in greater detail in Algorithm 3:

- Select candidate solutions $\mathbf{X}$,
- Evaluate the true function $f^\star$ at each position, producing a matrix of cost values $\mathbf{Y}$,
- Create a set of particles, each of which represents a function, and
- Calculate the likelihood of each particle (function) with respect to $\mathbf{X}$ and $\mathbf{Y}$.

Algorithm 3 Particle filter for one time step in the EO-DBN

```
# Create a population of $N_\mathbf{X}$ candidate locations and obtain values
$\mathbf{X} = (\mathbf{x}_1 \ldots \mathbf{x}_{N_\mathbf{X}})$, with $\mathbf{x}_i \sim \rho(\mathbf{x}^\star \,|\, f_i)$ and $f_i \sim \rho(f)$
$\mathbf{Y} = (\mathbf{y}_1 \ldots \mathbf{y}_{N_\mathbf{X}})^\top$, with $\mathbf{y}_i = f^\star(\mathbf{x}_i)$
# Create $N_\mathbf{P}$ particles and calculate the normalized likelihood for each
$\mathbf{P} = (f_1 \ldots f_{N_\mathbf{X}})^\top$, with $f_i \sim \rho(f)$
$\mathbf{L} = \Big(\mathcal{L}(f_1 \,|\, \mathbf{X},\mathbf{Y}) \ldots \mathcal{L}(f_{N_\mathbf{P}} \,|\, \mathbf{X},\mathbf{Y})\Big)^\top \big/ \sum_{i=1}^{N_\mathbf{P}} \mathcal{L}(f_i \,|\, \mathbf{X},\mathbf{Y})$ with $\mathcal{L}(f_i \,|\, \mathbf{X},\mathbf{Y}) = \prod_{j=1}^{N_\mathbf{X}} \rho(\mathbf{y}_j \,|\, f_i, \mathbf{x}_j)$
# We now have a discrete representation of the function posterior that may be sampled:
$\rho(f \,|\, \mathbf{X},\mathbf{Y}) := \mathbf{P}, \mathbf{L}$
```

The distinction between the true function ($f^\star$) and the function variable in the network (f) is subtle but important when describing the algorithm. The true function $f^\star$ is the real-world function that is the target of optimization; sampling it represents (potentially costly) function evaluations. In contrast, the variable f represents the algorithm's internal representation of its knowledge and assumptions about the true function. Ideally, sampling from a distribution over f will provide a good approximation to $f^\star$.

The issues of choosing a suitable prior $\rho(f)$, a meaningful sampling distribution $\rho(\mathbf{Y} \mid f, \mathbf{X})$, and an appropriate optimization distribution $\rho(\mathbf{x}^\star \mid f)$ are most easily addressed and understood by example. The next several sections will therefore be devoted to concrete demonstrations of how distributions are specified and what such specifications mean, beginning with toy examples and moving toward more practical and interesting ideas.

8.4 Example: Simple Parametric Priors

Distribution specification begins with a definition of the function class. Suppose, for example, that the function subject to optimization is known to be "similar to a cone". The prior $\rho(f)$ provides the specification of "a cone", the sampling distribution $\rho(\mathbf{Y} \mid f, \mathbf{X})$ provides a precise meaning for "similar to", and the optimization distribution $\rho(\mathbf{x}^\star \mid f)$ dictates how to extract the location of the global minimum from "a cone" as described in the prior. Consider, for example, the following distributions:

$$\rho(f) = \begin{cases} \rho(\mathbf{c}) & \text{if } f \in \{\lambda \mathbf{x}.g(\mathbf{x}, \mathbf{c}) \mid \forall \mathbf{x} \in \mathbb{R}^D . g(\mathbf{x}, \mathbf{c}) = \|\mathbf{x} - \mathbf{c}\|_2\} \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

$$\rho(\mathbf{c}) = N(\mathbf{0}, \Sigma_{\mathbf{c}}) \tag{8.3}$$

$$\rho(\mathbf{y} \mid f, \mathbf{x}) = N(f(\mathbf{x}), \sigma_{\mathbf{y}}) \tag{8.4}$$

$$\rho(\mathbf{x}^\star \mid f) = \delta(\mathbf{x}^\star - \mathbf{c}) \ . \tag{8.5}$$

The prior $\rho(f)$ uses lambda calculus to state that only cones of the form $\|\mathbf{x} - \mathbf{c}\|_2$ will initially receive any density. It does so by defining an auxiliary prior distribution over the cone's parameters, $\rho(\mathbf{c})$, which is in this case normal. In other words, $\rho(f)$ favors cones which are centered near the origin and gives no weight to functions which are not cones. While only one of the possible definitions of "a cone", this definition is precise.

The sampling distribution $\rho(\mathbf{Y} \mid f, \mathbf{X})$ also plays a significant role in the specification of the function class, providing a way of blurring the line between "cone" and "non-cone" by precisely defining the meaning of "similar to". In this particular instance, the distribution assumes that all true values are within a Gaussian sample of a cone's corresponding values. Interestingly, when coupled with the prior, this results in *every possible function* receiving at least some likelihood, since all values at all positions receive at least a small amount of nonzero density in the network.

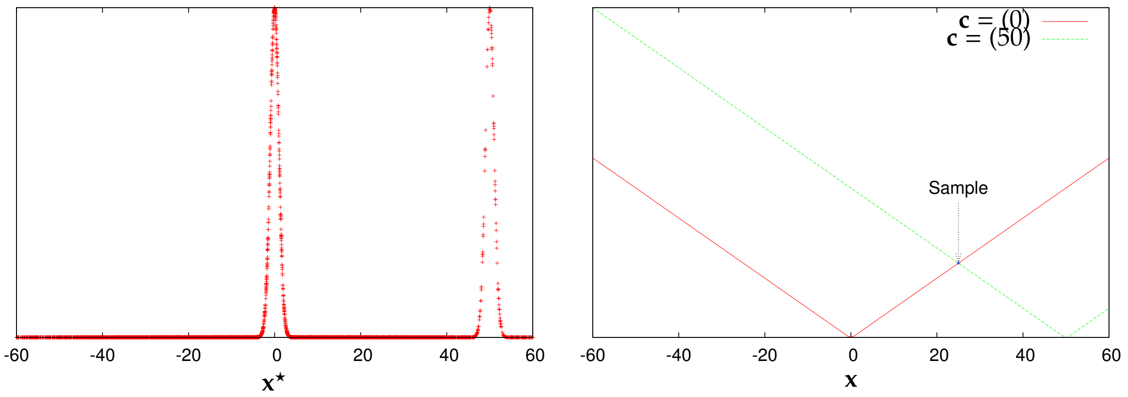
Finally, the optimization distribution $\rho(\mathbf{x}^\star \mid f)$ defines the goals or intent of the search process. Because of the way that "a cone" is defined, any particular cone sampled from the prior will have a minimum located at $\mathbf{c}$. The delta distribution is therefore used to indicate that the critical relationship is $\mathbf{x}^\star = \mathbf{c}$.

Together, these distributions represent an almost complete specification of optimization in the Bayesian model. All that remains is to choose a number of real function samples $N_\mathbf{X}$ and, because a particle filter will be employed, the number of particles $N_\mathbf{P}$. Of course, the various distributional parameters must also be specified. To lend concreteness to the discussion, consider the following settings:

$$\begin{aligned} D &= 1 & f^\star(\mathbf{x}) &= \|\mathbf{x} - 50\|_2 \\ N_\mathbf{P} &= 5000 & \Sigma_c &= \left((50/3)^2\right) \\ N_\mathbf{X} &= 1 & \sigma_\mathbf{y} &= 1 \ . \end{aligned}$$

In other words, 5000 particles will be used, a single sample taken, and the true function will be a one-dimensional cone centered at $\mathbf{x}^\star = 50$.

Following the steps of Algorithm 3, line 2 dictates that samples be taken from the function prior, then from the conditional optimization distribution, producing a



(a) Filtered posterior

(b) Possible functions

Figure 8.2: Cone prior with a true cone function

population of candidate solutions. In this example, exactly one sample is produced since $N_{\mathbf{X}} = 1$. Let the result of sampling from $\rho(\mathbf{c})$ be $\mathbf{c} = 25$; sampling from the trivial $\rho(\mathbf{x}^\star \mid f)$ therefore produces the (singleton) population of candidate solutions $\mathbf{X} = (25)$.

Line 3 of the algorithm describes the process of evaluating the true function $f^\star$ at all candidate locations, which in this case produces $\mathbf{Y} = (f^\star(25)) = (25)$. With a population of candidate solutions and their corresponding true values, it is now possible to perform inference; the observable variables are now assigned, so the particle filter may be applied. Line 5 describes the process of obtaining particles, producing a vector $\mathbf{P}$ of 5000 cones, each centered at a location drawn from $\rho(\mathbf{c})$. Finally, line 6 gives a formula for calculating the normalized likelihood of each particle from $\rho(\mathbf{Y} \mid f, \mathbf{X})$, producing an empirical approximation to the posterior $\rho(\mathbf{x}^\star \mid \mathbf{X}, \mathbf{Y})$ as shown in Figure 8.2(a).

With exactly one sample at the location 25, the two cones centered at 0 and 50 are maximally consistent with the data, as shown in Figure 8.2(b). The peak in Figure 8.2(a) corresponding to the true function's minimum at 50 is more sparsely represented than the wrong peak at 0 because $\rho(\mathbf{c})$ favors the origin, but the posterior $\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y})$ still provides good likelihoods for the correct minimum.

It is worth noting that had $\rho(\mathbf{Y} \,|\, f, \mathbf{X})$ not allowed for uncertainty or noise in its specification, it would have been practically impossible to obtain a useful distribution over $\mathbf{x}^\star$. In a continuous setting such as this, the likelihood of sampling the exact set of parameters which will produce perfectly consistent results is zero, and therefore some amount of uncertainty must be encoded in $\rho(\mathbf{Y} \,|\, f, \mathbf{X})$: a delta distribution does not provide sufficient approximate information.

Had the population of samples contained 2 locations with unique values, there would have been exactly one peak in the graph, since an additional evaluation would serve to differentiate between the two peaks. Significantly, even though the sample point is nowhere near the actual global minimum, it provides enough information to indicate where the minimum is. This is one important consequence of including a specification of the function class in the Bayesian optimization model; minima may be discovered without ever being members of the population of candidate solutions. The minima may be represented among the population of particles, but particle creation and evaluation do not rely on the direct evaluation of the true function $f^\star$; it is possible to discover the minimum without ever sampling it.

8.4.1 Problem Complexity in the Sampling Model

This example provides a good context for a discussion of the role of representation in problem complexity. Temporarily ignoring the complexity of particle filtering (it is an implementation detail independent of the target function), there are two major sources of problem complexity in the model: *sample complexity* and *computational complexity*.

Sample complexity is a notion researched extensively in the machine learning community, describing the number of samples of $f^\star$ required to obtain a good approximation to it. This is relevant in the context of the Bayesian optimization model because it casts part of the optimization problem as a learning problem, specifically the problem of taking sufficient samples of $f^\star$ so that a good approximation may be obtained; the approximation need not be perfect everywhere, but should be correct enough that accurate optimization may be performed. In this particular example, the space of learning hypotheses is the set of all possible centers for the cone defined in (8.2).

This example's representation is admittedly simplistic and limited, as well as being only one-dimensional. It is therefore easy to see that two unique samples are sufficient to determine the center of the true cone. One sample narrowed it down to basically two possible parameter vectors, and one more would narrow it further. In general, establishing bounds on the sample complexity of a particular representation is part of a large body of existing and ongoing machine learning research, but when an appropriate representation is used, upper and lower bounds on the number of required samples can be obtained [Mitchell 1997; Christianini and Shawe-Taylor 2000]. Specifically, if the representation's "Fat-Shattering Dimension" is discoverable and finite, its sample complexity may be bounded [Valiant 1984; Bartlett et al. 1996; Kearns and Schapire 1994], providing a *natural worst-case stopping criterion* for an evolutionary optimization algorithm. Although both upper and lower bounds are often available, the lower bound is not likely to be correct in the Bayesian optimization model because only enough of $f^\star$ to facilitate optimization must be approximated.

The computational complexity of a problem is evident in the definition of $\rho(\mathbf{x}^\star \mid f)$, a distribution which provides a statistical mapping between a function drawn from the prior and the location of its global minimum. In this example, it is computationally trivial to draw samples from $\rho(\mathbf{x}^\star \mid f)$ because it simply states that $\mathbf{x}^\star = \mathbf{c}$. Such simplicity is not the rule in general, however, and the complexity of sampling from that distribution is in many cases significantly higher. Consider, for example, the prior

$$\rho(f) = \begin{cases} \rho(\mathbf{c}) & \text{if } f \in \{\lambda \mathbf{x}.g(\mathbf{x}, \mathbf{c}) \mid \forall \mathbf{x} \in \mathbb{R}.g(\mathbf{x}, \mathbf{c}) = \mathrm{ANN}_1(\mathbf{x}, \mathbf{c})\} \\ 0 & \text{otherwise} \end{cases} \tag{8.6}$$

where ANN_1 is a multilayer feed-forward artificial neural network with some predefined structure. The parameter vector $\mathbf{c}$ in this case represents the weights on the network edges. If the network is large enough, it may be capable of immense flexibility. Even if it has a relatively low fat-shattering dimension and therefore has low sample complexity, there is another source of complexity in this representation: $\rho(\mathbf{x}^\star \mid f)$.

Artificial neural networks, for all of their strengths, do not easily lend themselves to output minimization. Therefore, $\rho(\mathbf{x}^\star \mid f)$ may actually hide *another optimization problem* with its own associated complexity! While in many cases this is unacceptable and a representation more amenable to minimum discovery should be used, in other cases it represents a reasonable tradeoff; in many real-world scenarios, e.g., tuning the parameters of an oil refinery to produce maximum output, computer time is extremely cheap compared to function evaluations, so offline optimization of an ANN is acceptable.

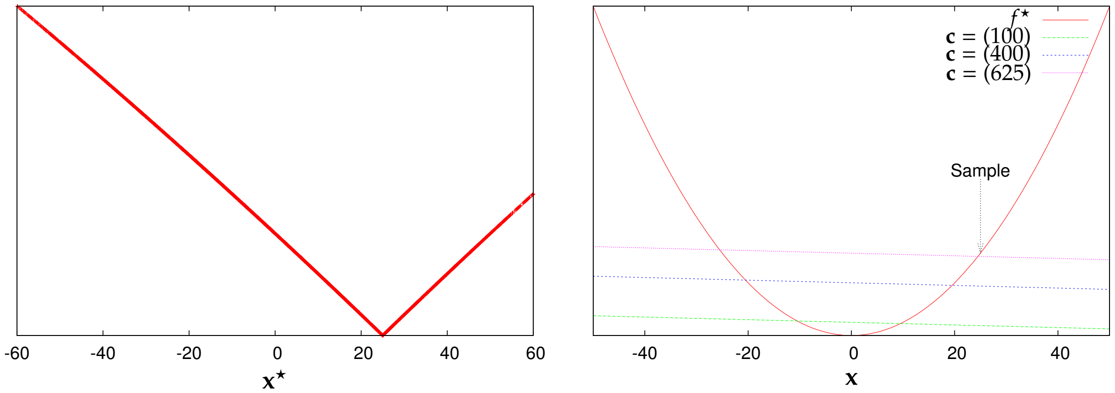
That the complexity of the problem is inherently tied to the flexibility of the representation of the prior serves to clarify part of No Free Lunch, especially the proof that problem difficulties cannot be ranked in the absence of a specific algorithm [Macready and Wolpert 1996]: it is not, in fact, the choice of *algorithm* that makes ranking possible among problems, but the choice of *representation*.

Additionally, the Bayesian optimization model provides a means by which the problem complexity can be quantified: it consists of an upper bound on the number of required function evaluations combined with the computational complexity of sampling from $\rho(\mathbf{x}^\star \mid f)$. That this complexity can be obtained at all is a strength of the model.

8.4.2 Wrong Priors

No Free Lunch dictates that any algorithm may be deceived, a difficulty to which the inference algorithm is not immune. Suppose, for example, that the true function is not a cone, but a parabola centered at the origin: $f^\star(\mathbf{x}) = ||\mathbf{x}||_2^2$. With all other details unchanged from the example, the new posterior distribution is shown in Figure 8.3(a) and is anything but reasonable. The actual location of the minimum does not receive the lowest likelihood, but it certainly does not receive the highest. Oddly enough, the likelihood is lowest at the sampled location and increases when moving away from it.

This behavior is explained by Figure 8.3(b), which shows that moving away from the sample, at least until a function is found which crosses through $(\mathbf{x}, \mathbf{y})$, causes the likelihood to increase. In this particular case, even adding more sample points fails to correct the distribution unless those points are generated very near the origin. Without the aid of the graphs, it may seem somewhat surprising that a conic prior does not allow the inference algorithm to perform well when given a parabolic function, but in reality parabolas receive vanishingly small likelihoods in the defined class. The class is “similar to a cone”, and the definition of “similar to” is “within a Gaussian sample of”, which a parabola clearly violates over most of its extent. Another, more useful definition of “similar to” will be given later that *does* admit the optimization of a parabola given a conic prior, but this particular definition of $\rho(\mathbf{Y} \mid f, \mathbf{X})$ does not.



(a) Likelihood

(b) Possible functions

Figure 8.3: Conic prior with a parabolic $f^\star$

It is of course, possible to slightly extend $\rho(f)$ so that it at least does something reasonable in a bounded region. One may, for example, add a scale parameter that allows the cone to become steep or shallow, allowing it to fit the area around the parabola's global minimum well enough to indicate its location. This approach, while useful, is a slippery slope: if an asymmetric function is targeted, should yet another parameter be added? What about bumpy functions? Each new parameter increases the sample complexity, often significantly, and such extensions warrant careful consideration.

8.5 Example: Discrete Priors

Perhaps the simplest way to specify $\rho(f)$ is as a discrete set of functions with associated fixed probabilities:

$$\rho(f) = \begin{cases} p_1 & \text{if } f = \lambda \mathbf{x}.f_1(\mathbf{x}) \\ p_2 & \text{if } f = \lambda \mathbf{x}.f_2(\mathbf{x}) \\ \vdots & \\ p_N & \text{if } f = \lambda \mathbf{x}.f_N(\mathbf{x}) \\ p_0 & \text{otherwise} \end{cases} . \tag{8.7}$$

So long as $\forall i.p_i \geq 0$ and $\sum_{i=0}^{N} p_i = 1$, any set of probabilities and any number or variety of functions may be specified. Once again, sampling from $\rho(f)$ produces a function rather than a scalar, the notation of lambda calculus is employed in the specification.

In this case, the prior makes the function class of interest very clear: the true function is assumed to be one of those listed. Whether this prior is reasonable or not depends on what an optimization practitioner knows about his problem; if he knows that he is optimizing one of the listed functions, it is a useful prior that will provide good information, otherwise it is not.

One can conceive of a situation in which a conference deadline is approaching and an algorithm which performs well on a number of benchmarks is needed.

In this case, the prior distribution might be defined as

$$\rho(f) = \begin{cases} \frac{1}{3} & \text{if } f = \lambda \mathbf{x}.f_1(\mathbf{x}) \text{ where } f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2 \\ \frac{1}{3} & \text{if } f = \lambda \mathbf{x}.f_2(\mathbf{x}) \text{ where } f_2(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 10\sum_{i=1}^{D} 1 - \cos(2\pi x_i) \\ \frac{1}{3} & \text{if } f = \lambda \mathbf{x}.f_3(\mathbf{x}) \text{ where } f_3(\mathbf{x}) = 20 + e - 20\exp\left(\frac{-\|\mathbf{x}\|_2}{5\sqrt{D}}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right) \\ 0 & \text{otherwise} \end{cases} \tag{8.8}$$

where D is the dimensionality (which in this section will be assumed to be 1) and the functions are the well-known Sphere, Rastrigin, and Ackley, respectively. Because the minimum of each of these functions is at the origin, the optimization distribution is

$$\rho(\mathbf{x}^\star \mid f) = \delta(\mathbf{x}^\star) \ . \tag{8.9}$$

Additionally, the sampling distribution can dictate that values must precisely match function output

$$\rho(\mathbf{Y} \mid f, \mathbf{X}) = \prod_i \delta(\mathbf{y}_i - f(\mathbf{x}_i)) \tag{8.10}$$

making the function class “precisely one of the listed benchmarks”.

It is relatively easy to see that Algorithm 3 will only generate candidate solutions at the origin, since all functions sampled from $\rho(f)$ have their minimum there. Because of this, the algorithm will fail to distinguish between functions, but will still readily find the global minimum when presented with a function represented in the prior. In fact *no function evaluations are needed* to find a correct posterior, since the distribution $\rho(\mathbf{x}^\star \mid f)$ as defined in (8.9) is not actually conditioned on the function! The posterior will immediately put all weight on the location **0**, which is, in fact, the global minimum for all functions in this class.

What happens if the true function is not listed in the discrete prior? Unless the true function either has its global minimum at the origin like those listed or the unknown case is assigned non-zero probability, the likelihood normalization step of the algorithm will divide by zero and fail. This is reasonable behavior because the specification is too limited; the algorithm simply performs as instructed, and fails to provide any information at all if none can be obtained from the given specifications.

8.5.1 No Free Lunch

It is true that this example is trivial, not likely to happen in the real world, and even somewhat silly. It serves, however, to illustrate an important point: the inference algorithm will take advantage of all of the information available to it. If it is, in fact, known that $f^\star$ is one of several that can be finitely listed, then the algorithm will select among them as quickly as their similarities will allow, thus efficiently providing the location of the global minimum. If, however, a far more flexible representation is needed and the function class is greatly expanded, it is possible that many samples will be required to find the global minimum.

One can think of the extreme case where the class of all possible functions is (more or less) equally represented in the distribution specifications. Because arbitrary point-value pairs are allowed in such a class, it must have an uncountable representation: a single function is an uncountable list of such pairs, and there are uncountably many such functions.

Even if it were possible to assign equal non-zero weight to an uncountable number of functions, which it is not, no amount of sampling would serve to distinguish between them or to give any inclination as to where the global minimum is located; sampling a single point produces a single value, but even points arbitrarily close to the recent sample remain unknown. The situation only worsens in the presence of noise.

In other words, there is no algorithm that can perform better than random search in consideration of the class of all functions. This result is simply a restatement of the well-known No Free Lunch theorem for optimization [Wolpert and Macready 1997]. While the theorem was proved for discrete problems, the Bayesian optimization model makes it obvious in the context of continuous problems as well; any prior that represents all possible functions equally (if such a prior could even be defined in a continuous setting) cannot allow any information to be gained through sampling, and therefore no algorithm will do better than random search.

The solution is to specify a more limited class than that containing all possible functions, one that is useful *for the problem at hand*. In this toy example, the problem is to do well on one of several benchmark functions; the class is extremely limited and arguably not at all useful in real-world settings, but it performs perfectly in the given situation. In fact, compared to any other algorithm, it is by far the best performer on the three listed benchmarks, since it requires absolutely no function evaluations to do its job. When compared to other algorithms on functions whose global minimum is not precisely at the origin, however, it will clearly be a very poor performer.

Because No Free Lunch is a well-known concept, researchers are increasingly interested in finding function classes that are well matched to their algorithms. The determination of an algorithm–class pairing is generally approximated by testing the algorithm on one of a number of benchmark functions, then generalizing the results by distilling various defining characteristics of those benchmarks. It has been argued that this is a somewhat fruitless enterprise because it actually fails to answer the pairing question [Whitley and Watson 2006], but it has been the only generally-applicable approach available. The Bayesian optimization model and this last example serve to clarify the reasons for this: every algorithm uses some amount of domain knowledge, and the amount of knowledge incorporated is generally hidden within the machinery of the algorithm. It is rarely clear exactly what function class assumptions are encoded in algorithms such as PSO, GAs, EDAs, etc., but such assumptions are undeniably present, else No Free Lunch demands that the algorithms do no better than random search.

The need to answer the algorithm–class pairing question is therefore at the heart of the consequences of No Free Lunch; nothing can do better on average than random search when considering all functions, but most algorithms are well-suited for at least a useful subset of those functions. Optimization methods should therefore be *designed* for a particular function class, not merely studied to *discover* that class after their creation.

8.6 Example: More General Priors

Prevalent in optimization algorithms is the assumption that hill-climbing is a useful behavior. This assumes, in essence, that the portion of the function which contains the global minimum is "similar to a bowl". In fact, this assumption is buried in a number of evolutionary optimization algorithms, many of which tend to be good hill-climbers.

Particles in PSO, for example, take their current velocity into account when moving, indicating a preference for continuing in what was previously a good direction. This greedy behavior, while balanced by new information, is an attempt at hill-climbing, as is the fact that only the best solutions in a neighborhood are considered. GAs and EDAs alike perform selection before producing new candidate solutions, effectively treating poor performers in the population as less useful than other members; the minimum is assumed to be near other locations of low value and evolutionary operators push a population toward those values. These assumptions of self-similarity and hill-climbing generally point to a function that is, at least at some scale or in the region of interest, "similar to a bowl".

There is nothing wrong with these assumptions, provided that they represent a desirable class of functions. The problem lies in the way that "similar to a bowl" is traditionally defined: implicitly and as part of the subtle machinery of the algorithm. The Bayesian optimization model admits more direct encoding of phrases such as "similar to a bowl", and the example that follows provides a definition that captures the essence of that description. As previously seen, the combination of $\rho(f)$ and $\rho(\mathbf{Y} \,|\, f, \mathbf{X})$ can specify the function class in an intuitive way.

Many functions fit a human definition of "bowl-shaped", including parabolas, cones, and functions describing distinctly non-parabolic, real-world dinnerware. What, then, is the essence of the definition? Consider the following distributions:

$$\rho(f) = \begin{cases} \rho(\mathbf{c}) & \text{if } f \in \{\lambda \mathbf{x}.g(\mathbf{x}, \mathbf{c}) \mid \forall \mathbf{x} \in \mathbb{R}^D.g(\mathbf{x}, \mathbf{c}) = \|\mathbf{x} - \mathbf{c}\|_2\} \\ 0 & \text{otherwise} \end{cases} \tag{8.11}$$

$$\rho(\mathbf{c}) = N(\mathbf{0}, \Sigma_{\mathbf{c}}) \tag{8.12}$$

$$\rho(\mathbf{Y} \,|\, f, \mathbf{X}) \propto \prod_{i<j} p(\mathbf{y}_i, \mathbf{y}_j \,|\, f, \mathbf{x}_i, \mathbf{x}_j) \tag{8.13}$$

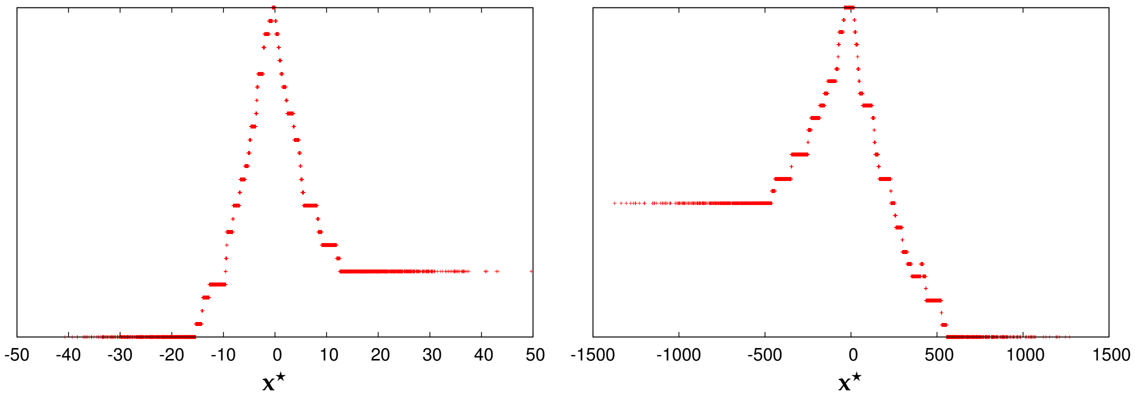
$$p(\mathbf{y}_1, \mathbf{y}_2 \,|\, f, \mathbf{x}_1, \mathbf{x}_2) = \begin{cases} 1 - \alpha & \text{if } \operatorname{sign}(\mathbf{y}_1 - \mathbf{y}_2) = \operatorname{sign}(f(\mathbf{x}_1) - f(\mathbf{x}_2)) \\ \alpha \in (0, 1) & \text{otherwise} \end{cases} \tag{8.14}$$

$$\rho(\mathbf{x}^\star \,|\, f) = \delta(\mathbf{x}^\star - \mathbf{c}) \ . \tag{8.15}$$

The function prior, defined in (8.11) and (8.12), is the familiar conic prior from Section 8.4, and the optimization distribution $\rho(\mathbf{x}^\star \mid f)$ is also familiar. These definitions therefore indicate that “a bowl” is defined to be “a cone” as specified in a previous example.

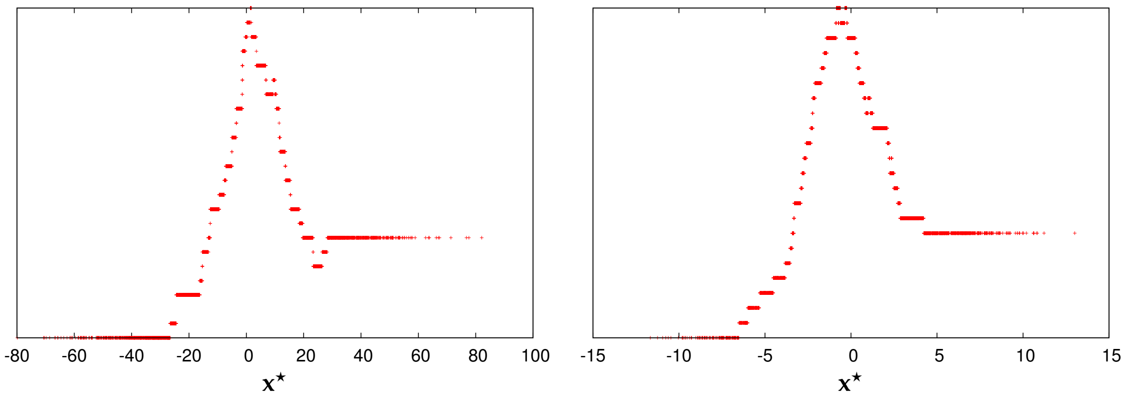
The distribution that sets this example apart from the others is $\rho(\mathbf{Y} \mid f, \mathbf{X})$, defined in (8.13) and (8.14). Its definition captures the meaning of “similar to”, a significant part of the specification of the function class. In previous examples, this distribution was based on the notion that $\mathbf{y} = f(\mathbf{x})$, sometimes with added white noise. In this example, the distribution is defined based on the *pairwise relationships* between values, not the values themselves; the probability of observing a particular value relationship is higher with a consistent “bowl” than with one whose corresponding pairwise relationship is inconsistent. In all examples that follow, $\alpha = 0.475$, effectively assigning a generous amount of probability to bowls as defined in (8.11) that are inconsistent with a particular observed value pair. A value so close to 0.5 indicates that a great deal of noise will be tolerated in the function: so long as the majority of pairs point toward the global minimum, the majority need not be a large one.

The definition of bowl-shaped using these distributions is actually quite intuitive: given two points, the point with the lower value will tend to be closest to the minimum. Any function which, like a cone, exhibits a general trend toward the global minimum is a member of this class. Even very noisy functions can be members of the class, provided that random sample pairs that point toward the global minimum tend to outnumber those that do not. Thus, “similar to” means “having the same basic trend, as manifested by random sample pairs”, and captures the essence of a human definition of a bowl far more adequately than the example in Section 8.4.



(a) DeJongF4

(b) Griewank



(c) Ackley

(d) Rastrigin

Figure 8.4: One-step likelihood on common one-dimensional benchmarks

Obviously, this function class definition requires that $N_{\mathbf{X}} \geq 2$. The application of this class definition is no different than for any of the previous examples; Algorithm 3 still applies directly. The results of its application to several common benchmark functions in 1 dimension are shown in Figure 8.4. The following parameters were used to generate the graphs: $N_{\mathbf{X}} = 10$, and $N_{\mathbf{P}} = 5000$.

The behavior of this algorithm given the bowl-shaped prior defined above is notable: it not only behaves rationally for DeJongF4, which while being bowl-shaped is very different from a cone, but also for bumpy functions such as Griewank, Ackley, and Rastrigin, verifying the claim that even noisy functions can be part of the class, provided that they display a general trend toward the global minimum.

8.6.1 The Burden of Specification

Let us take a short digression and consider Particle Swarm Optimization. The motivation behind the algorithm was a simulation of natural flocking behavior, and by paring down the math, PSO was born [Kennedy and Eberhart 1995]. This discovery was somewhat serendipitous and accidental, but it has proven to be a fruitful area of research since its discovery. In the design of PSO, with the exception of some discussion about specific functions, no direct consideration was made of the *class* of functions on which it may be expected to work well. In fact, much of the research following its discovery has been focused on evaluating its performance in various applications, such as the training of artificial neural networks and support vector machines [Eberhart and Shi 2001; Mendes et al. 2002; Paquet and Engelbrecht 2003b].

While the pursuit of interesting applications of PSO is valid and useful, it is also telling; here is an algorithm that appears to have promise, but exactly what kind of promise it has is unclear. Researchers must simply test it in various settings and report their findings. To be fair, PSO is not at all unique in this sense; establishing an algorithm–class pairing is approached in this manner for practically every evolutionary optimization algorithm.

In contrast, the Bayesian optimization model does not admit such serendipitous algorithm design; it requires a complete, *a priori* specification of the class of functions on which it is expected to work, and on that class it will work as efficiently as it can; anything less will not produce a working algorithm. While this appears to be a solution to the algorithm–class pairing problem, it may seem as though a greater burden has been placed on the algorithm designer or optimization practitioner: one of complete and precise specification of intent.

This is indeed a tradeoff, but it is well worth the cost. Rather than producing an algorithm based heavily on inspiration or intuition, one may now take control of how optimization will proceed based on actual available domain knowledge. When such knowledge is present, it may (indeed, it must) be incorporated directly into the specification of the algorithm, which then makes effective use of all of the information it receives. This approach to algorithm design is certainly no worse than that traditionally used, and is in fact a good deal better.

A somewhat more compelling argument against the Bayesian optimization model can be made when little or no domain knowledge is available. In these settings, practitioners commonly employ several of the algorithms at their disposal simultaneously in an attempt to find one that works well. That the inference algorithm requires explicit *a priori* specification of intent appears to make it a poor choice in these situations because the real intent is not known. This is, however, not the case.

In light of No Free Lunch, a choice between available optimization algorithms (e.g., PSO, GAs, Simulated Annealing, Amoeba Search, etc.) is really just a choice between predefined function classes. The Bayesian optimization model does

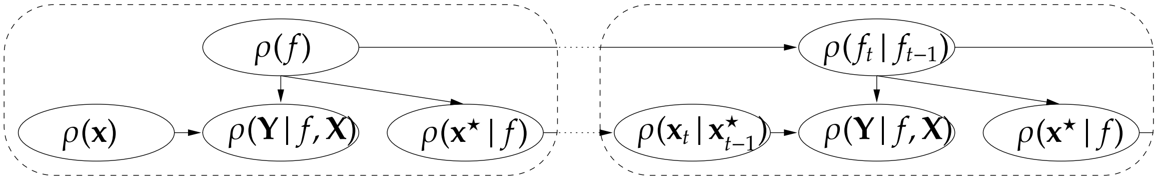

$\rho(f)$
$\rho(f_t \mid f_{t-1})$
$\rho(\mathbf{x})$
$\rho(\mathbf{Y} \mid f, \mathbf{X})$
$\rho(\mathbf{x}^\star \mid f)$
$\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$
$\rho(\mathbf{Y} \mid f, \mathbf{X})$
$\rho(\mathbf{x}^\star \mid f)$

Figure 8.5: The EO-DBN

not take that choice away, it simply makes it clear. In fact, the Bayesian optimization model's inference algorithm has the potential to *simplify* the process of finding the right algorithm for the job: the algorithm itself (inference) remains fixed, but one may explore the nature of the application at hand by observing the performance of various function class definitions; instead of choosing among several opaque algorithms, one may select transparent specifications whose application serves to narrow down the true function class.

Therefore, while it is true that a greater burden of explicit specification has been placed on the practitioner than exists in more traditional contexts, the equally heavy burden of blindly searching for the right tool has been lifted.

8.7 Introducing Evolution: The Evolutionary Optimization DBN

The discussion has been focused thus far on a static Bayesian optimization model that makes use of a single population of samples. The success of evolutionary optimization methods suggests that something may be gained by using multiple generations of samples over time, and the incorporation of that idea produces a more complete model of optimization based on Dynamic Bayesian Networks (DBNs): the Evolutionary Optimization DBN (EO-DBN).

The dynamic model that incorporates evolution is shown in Figure 8.5 and is inspired by EDAs. The figure displays the distributions instead of the variables, but the meaning should be clear: the variable over which a distribution is defined is the variable belonging in the node, and all conditional variables are parents of that node. The inference algorithm's task is to obtain and use the distributions shown.

This perspective introduces the distributions $\rho(f_t \mid f_{t-1})$ and $\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$. The former defines a way in which functions may change over time, and the latter defines how each new population is generated from the last. A closely related distribution, $\rho(\mathbf{x})$, is used for generating the initial population. Though the previous examples have used $\rho(f)$ and $\rho(\mathbf{x}^\star \mid f)$ to generate the initial population, this is not required in general and the greater flexibility of a separate $\rho(\mathbf{x})$ can be useful.

There are two basic types of evolution introduced by this model, and both will be addressed here. The first is *population evolution*, which is what is normally considered to be the meaning of the term "evolution" in evolutionary algorithms. The second is *artificial evolution*, which is an interesting statistical trick (with an unfortunately overloaded name) used to introduce needed diversity into empirical distributions produced by numerical inference methods such as particle filters.

8.7.1 Population Evolution

Population evolution is widely considered to be the key element of evolutionary algorithms. It is the process of generating a population using information gained from evaluating a previous population. In a more general sense, it is actually better described as "population replacement", since the current members of a population need not have a direct relationship to members of the previous population, as is the case with EDAs.

An important motivation for the introduction of population evolution into the Bayesian optimization model is the ability to work with functions at varying levels of detail. Consider Rastrigin, which at very low or "blurry" levels of detail looks smooth and bowl-shaped, but at higher levels of detail appears very bumpy. A posterior created by sampling from a low-confidence (high variance) function prior may point out roughly where the global minimum is located, but in reality the neighborhood given high likelihood in the posterior needs to be searched in greater detail to find the true minimum. Once all samples begin to fall within the same local minimum, that area of the function is again smooth and bowl-shaped and can be explored in greater detail. Of course, in any continuous setting the goal is not to find the minimum, but to close in on it, since the probability of actually sampling the global minimum is zero; there are simply too many neighboring points that may be arbitrarily close in value or position.

This reduction in scale frequently cannot occur if all of the information gained about the function is used all of the time, since previous information about the larger function may naturally include other local minima and will therefore have the potential to deceive the algorithm while it explores the current area of interest. Thus, selectively discarding or discounting information within members of the previous population allows more efficient exploration to occur in the future, providing a strong argument in favor of population evolution.

Additionally, population evolution causes inferred posterior distributions to become increasingly narrow, providing a natural stopping criterion based on the desired level of detail: stop when the high-likelihood values cover a sufficiently small area.

The exact nature of population evolution is defined in $\rho(\mathbf{x}_t \,|\, \mathbf{x}^{\star}_{t-1})$, a distribution that provides another opportunity to specify an algorithm designer's intent and thus the class of functions for which the algorithm is well-suited. The kind of population evolution described above assumes something that may or may not be true: that past information provides indicators for how to productively narrow the search. The purpose of this discussion is not so much to indicate how population evolution should occur, but rather that it is directly specified in the model. This is in keeping with the theme of this entire work: decisions must undoubtedly be made during the specification of an optimization algorithm; the key is to make those decisions clear and explicit.

8.7.2 Artificial Evolution

"Artificial evolution" was actually produced in the context of dynamic Bayesian networks, and not in the context of traditional evolutionary algorithms. It refers to a statistical trick that allows smoothing of the posterior distribution to occur via artificially noisy transitions [Liu and West 2001].

While $\rho(f_t \mid f_{t-1})$ can obviously be used to model a truly dynamic target function, it can also be used to model *changing uncertainty* about a static function. It can, of course, combine the two and model both at once. For truly static functions, however, it is tempting to define the distribution thus:

$$\rho(f_t \mid f_{t-1}) = \begin{cases} 1 & \text{if } f_{t-1} = f_t \\ 0 & \text{otherwise} \end{cases} \tag{8.16}$$

Given that the EO-DBN generally requires a numerical method of inference, this distribution is not likely to be very useful. Consider that the estimate of the distribution over functions at time $t-1$ is actually the posterior $\rho_{t-1}(f \mid \mathbf{X}, \mathbf{Y})$, which is represented by a discrete set of particles and associated likelihoods. Sampling from such a discrete distribution is a solved problem [Liu and West 2001], but over time the number of unique samples will decrease, since (8.16) displays no uncertainty whatsoever that what it sees is still correct; no noise is introduced that would allow other "nearby" particles to be generated.

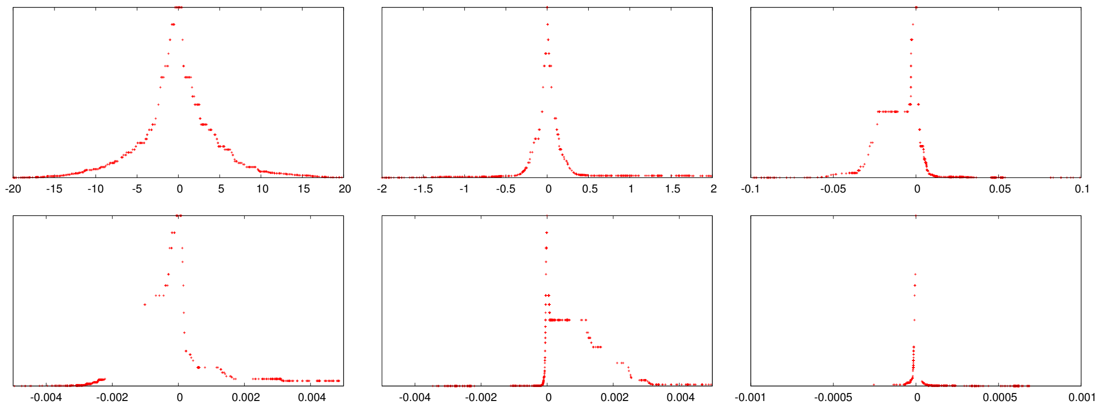
A common solution for this problem is to make use of "artificial evolution", which basically changes $\rho(f_t \mid f_{t-1})$ to introduce noise (often additive Gaussian) into the system, allowing particles to be created that have not been sampled previously.

As mentioned before, this approach is equivalent to the more principled idea of applying kernel smoothing to the posterior before resampling [Liu and West 2001], but is more popular because it is simpler to implement. The same argument applies to $\rho(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y})$, since it is also a discrete approximation of a continuous distribution.

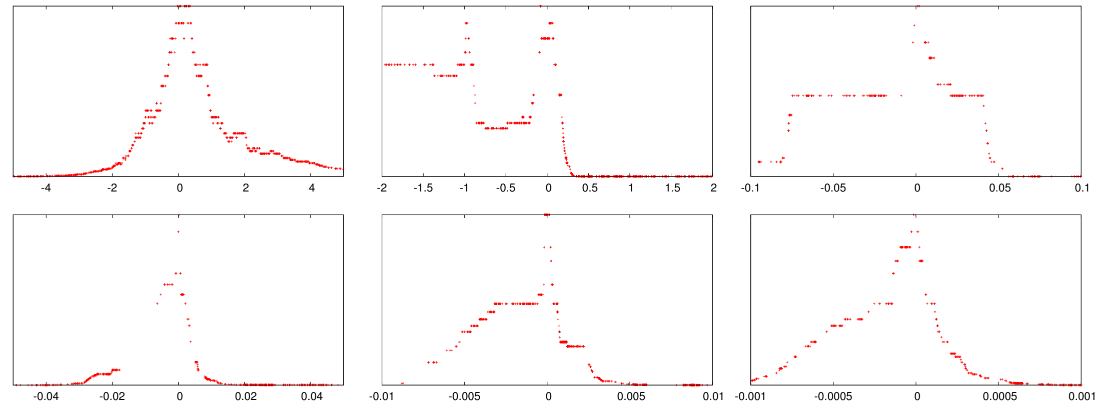
The definition of $\rho(f_t \,|\, f_{t-1})$, because it allows for the optimization of dynamic functions, obviously represents an important part of the function class specification. What is perhaps not as obvious is that introducing artificial evolution to that distribution also subtly affects the class specification. For example, if Gaussian noise is employed, there is an implicit assumption of locality. Care should therefore be taken that any noise added to either $\rho(f_t \,|\, f_{t-1})$ or $\rho(\mathbf{x}_t \,|\, \mathbf{x}^\star_{t-1})$ is consistent with the specified function class. In the examples that follow, the locality assumption is assumed to be valid and $\rho(f_t \,|\, f_{t-1})$ and $\rho(\mathbf{x}_t \,|\, \mathbf{x}^\star_{t-1})$ are assumed to be delta distributions with additive Gaussian noise.

Once it is established that Gaussian locality is reasonable in these distributions, it is not clear exactly what variance the noise should have. One sensible approach is to let the variance for each particle be dependent upon the Euclidean distance to its nearest neighbor. This approach is employed in several tests of the evolutionary algorithm, specifically with the standard deviation in each dimension equal to twice the distance to the nearest neighbor. Results of applying this technique to the "bowl-shaped" class with $N_{\mathbf{X}} = 20$ and $N_{\mathbf{P}} = 1000$ are shown in Figure 8.6.

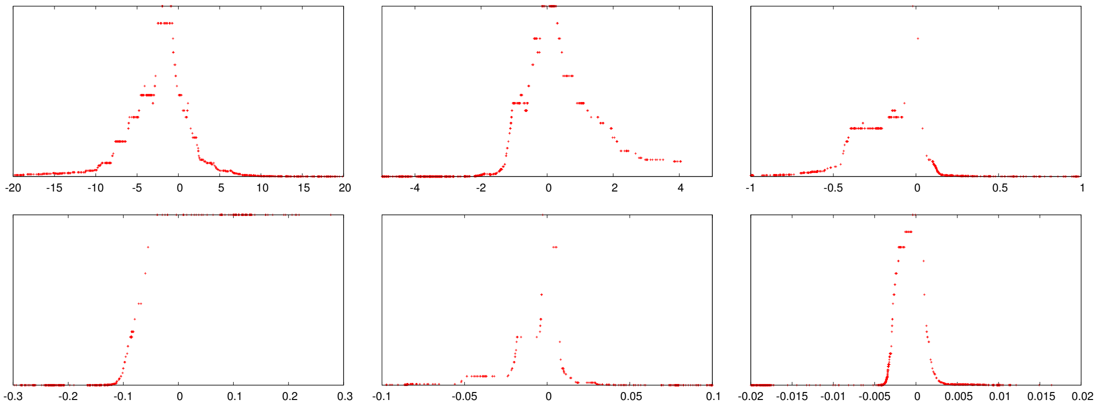
The distributions should be read from left to right, top to bottom, with each new distribution representing one completed generation. As expected, the variance is fairly high early in the run, then (taking note of the scales on the graphs) it settles down into a high-resolution area of the function's domain. While the graphs shown are for one-dimensional functions, it has been verified that this method behaves similarly in spaces of higher dimensionality.



(a) Sphere



(b) Rastrigin



(c) Ackley

Figure 8.6: March of posteriors over $\mathbf{x}^{\star}$ for one-dimensional functions in the EO-DBN

The graphs have some interesting characteristics that are worthy of mention. In Figure 8.6(c), the fourth generation exhibits some strange behavior: the distribution has a high, flat plateau on one side. This occurs with the "bowl-shaped" specification when all or most of the function evaluations are on one side of the minimum. In that context, all pairs point in the same direction, and therefore the minimum may be anywhere on one side. This does not happen often, but when it does, the behavior is that depicted. Similar but less striking examples of this behavior appear in Figures 8.6(a) and 8.6(b), as well. Of additional interest are the early generations for Rastrigin, where the multimodality of the function is in evidenced in the distributions.

8.7.3 The Curse of Dimensionality

EO-DBN does not claim to produce a free lunch, nor does it claim to overcome the curse of dimensionality; as the dimensionality of a problem increases, so does the size of the search space, independent of the applied optimization method.

Algorithms that address the curse of dimensionality generally do so by employing heuristics that artificially limit the search space, focusing attention on regions that are the most promising, contain the most information, or conform to other such metrics of usefulness. Some limit the space by discovering low-dimensional manifolds [Saul and Roweis 2003] while others make use of problem constraints, but in the end all of them are using some amount of domain knowledge to limit the scope of their efforts. In the presence of a multidimensional uncountable space, indeed nothing else can be done.

The EO-DBN, while not claiming to overcome this curse of dimensionality, does allow domain knowledge to be incorporated in a principled way and reduces the search space over time by focusing its attention on the most promising areas of the space, where "promising" is defined by the function class specification, the declaration of search intent, the definition of population evolution, and the target function itself. It may be viewed as an adaptive approach to space reduction that, while not overcoming the curse of dimensionality, employs all domain information available to limit the scope of search as efficiently as it can.

Nowhere is the curse of dimensionality more evident in the EO-DBN than in the use of the particle filter as a means of approximating the posterior distributions, although it is also present in the sample complexity. The particle filter, while easy to implement and straightforward to explain, is not necessarily the best choice in these situations because it can require a large number of particles to provide a good approximation, especially as dimensionality increases. It is true that these particles do not represent function evaluations and may therefore incur negligible cost in some optimization problems, but in many cases the problem is entirely contained in simulation (e.g., computer vision problems) and every sample costs roughly the same amount, be it a sample of the target function or a sample from a prior.

This brings up two interesting points. First, other numerical inference methods exist and may be more efficient than particle filters for a given situation. For example, Markov Chain Monte Carlo simulation can be effective in some situations [Russel and Norvig 2003], an interesting avenue for future research.

Second, if the bottleneck of running the EO-DBN algorithm is in the numerical inference method, it is likely that either the representation of the prior is overly complex or that function evaluations are very inexpensive. If the prior admits only an overly complex $\rho(\mathbf{x}^\star \mid f)$ that cannot reasonably be made simpler, or if $\rho(f)$ encodes a very large function class for which there is not sufficient information to reduce it, then the optimization practitioner has run headlong into No Free Lunch; either more information is needed to reduce the class size or the problem at hand (e.g., needle-in-a-haystack) is inherently difficult.

On the other hand, if function evaluations are cheap then it may make sense to work with larger populations of candidate solutions and smaller populations of particles. Inexpensive function evaluations translate to inexpensive information; in order to gain more from less frequent inference, information should be gathered more abundantly. Similarly, if more information is available, less statistical inference may be needed to obtain useful distributions in the first place. A tradeoff certainly exists in these situations, and a decision must be made by the optimization practitioner.

This again illustrates the fundamental purpose of the EO-DBN: to provide expressive decision-making power to the practitioner so that optimization can be as well-informed as possible. That these kinds of issues can be discussed and debated at all is largely due to the fact that the EO-DBN serves to clarify and expose them.

8.8 Discussion

Throughout this work and in the context of each example, several concepts have been discussed. These included the role and definition of problem complexity, the full expression of No Free Lunch and its consequences, the nature of the burden placed on the optimization practitioner, and the curse of dimensionality. This section is devoted to insights and observations that either have not been discussed previously or have not yet received sufficient attention.

8.8.1 Post-Design Empiricism

It should be now clear that empirical results showing how EO-DBN compares with other continuous optimization algorithms are absent from this work. This is not an oversight, but a purposeful omission.

There are basically two common reasons behind performing comparative experiments and using them to draw conclusions at the end of a work on optimization:

- To discover the function class to which the proposed algorithm is well-tuned, and
- To advocate the use of the algorithm on a (hopefully related) set of problems.

The first is easily addressed because the function class is *specified* in the EO-DBN rather than merely analyzed; it is a fundamental requirement of the algorithm.

The second point is more interesting because it is addressed by the overarching purpose of this work. The EO-DBN is a useful and instructive way of thinking about the problem of optimization, one consequence of which is an inference-based algorithm which makes the best use it can of the information it has been given.

Especially in Section 8.5, where a discrete prior was described, two informational extremes were explored. The first represents an enormous amount of domain knowledge, straightforwardly encoded in the prior and sampling distributions. In that case, *not one function evaluation* was required to obtain the global minimum: it was known to be at the origin through simple analysis of $\rho(\mathbf{x}^\star \mid f)$. The other extreme case was discussed in the same section in the context of No Free Lunch; when no information whatsoever is available, the algorithm never gains any more information from sampling: not even *countably infinite function evaluations* can help to find the global minimum.

The EO-DBN will therefore make use of all of the information at its disposal. This is true of more traditional approaches such as PSO and GAs, but in those cases the meaning and amount of embedded domain knowledge (encoded indirectly within the machinery of the algorithm) is unclear. In order to fairly compare the EO-DBN against any other algorithm, a fundamental question must first be answered: with how much and what kind of information should the EO-DBN be provided? If a large amount of information is made available (e.g., the discrete prior in Section 8.5), the algorithm will do very well, and if very little is provided, it will perform in precisely the manner that No Free Lunch dictates: no better than random search. In fact, the algorithm gracefully and automatically degrades to random search as information becomes more scarce.

Most black-box evolutionary optimization algorithms fall somewhere between these two extremes, and we have provided a description of bowl-shaped function class which is also somewhere in the middle. Other such descriptions are certain to exist and may be the subject of interesting future research. In the end, the EO-DBN not only provides a new way of thinking about algorithm design, it suggests a different perspective on the problem of optimization itself.

8.8.2 Model Expressiveness

After tackling static, noise-free, single-objective optimization, evolutionary algorithms are often retro-fitted to handle dynamic, noisy, and/or multi-objective optimization problems. Some solutions are more elegant than others, but it is frequently the case that they are applied after the fact.

In the case of the EO-DBN, however, the situation is different. Because it provides a statistical model of the sampling process and clear connections between function definitions that may change over time, the inclusion of sample noise and dynamic functions is straightforward. In addition, because the relationship of f to $\mathbf{x}^\star$ is a statistical distribution, such a distribution can easily be multimodal, allowing multiple minima to be tracked simultaneously[1].

---

[1] Note that this does not imply that all meanings of *multi-objective* can be easily incorporated. In particular, social welfare problems continue to be difficult regardless of the applied algorithm because of the fundamental impossibility of fairly assigning global utility given multiple individual utilities [Arrow 1950]. In these cases algorithms are often adapted to search for the Pareto Front, which is itself a problem of tracking infinitely many minima simultaneously.

8.8.3 Prior Information

The EO-DBN algorithm is fixed: statistical inference. The application of that algorithm to a particular problem therefore chiefly requires a specification of practitioner intent, which is encoded in various probability distributions in a clear and direct manner. The distributions $\rho(f)$, $\rho(\mathbf{Y} \mid f, \mathbf{X})$, and $\rho(f_t \mid f_{t-1})$ define the function class (the latter may be used to indicate a dynamic function or to incorporate artificial evolution); $\rho(\mathbf{x}^\star \mid f)$ defines the overall goal of search (which may not actually be optimization, but something different or more general); and $\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$ defines the way that evolution should operate.

The distribution $\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$ and its companion prior $\rho(\mathbf{x})$ are somewhat problematic. While the other distributions have demonstrably clear meanings, the significance of these two distributions and the consequences of their specification are less clear; they represent such indirect and fuzzy notions as “scale reduction” or “algorithm greediness”. These ideas are interesting, and some useful things can be said about them, but their meaning is not as directly nor clearly expressed as the others.

The reason for this relative lack of clarity in the presence of evolution is simple: while there are good reasons for introducing evolutionary techniques into the algorithm (besides dynamic functions), none of these reasons is explicitly stated in the definition of $\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$ and $\rho(\mathbf{x})$. Why, for example, must $\rho(\mathbf{x}_t \mid \mathbf{x}^\star_{t-1})$ exist in the first place? Could not some other relationship be used, perhaps one that does not assume that all of the useful information for one population comes from a distribution over the global minimum? The choice made here is essentially greedy, indicating that points likely to be near the global minimum have the most *value* in the optimization process. In other words, those points are implicitly assigned a higher *utility* by this choice of distribution.

The spirit of the EO-DBN is one of taking implicit things and making them explicit. This has been a successful venture except in this instance, where utility is implied by part of the network structure and its corresponding distribution definitions. The introduction of explicit notions of utility can allow for a more direct specification of the intent of the evolutionary process and is the subject of ongoing and future research.

8.9 Conclusions

The EO-DBN and the static Bayesian optimization model within it are useful ways of approaching the problem of continuous optimization. They provide a clear and direct means by which the function class of interest may be specified, they help to clarify the role of NFL in continuous problem complexity, and (with the exception of particle filter parameters and the choice of population size) the corresponding inference algorithm requires only a specific distribution-based declaration of intent for its operation.

While useful on these merits alone, the EO-DBN also points to some interesting avenues for future research, including a more thorough treatment of problem complexity (particularly sample complexity), the creation of a toolbox of useful function class definitions, the study of natural stopping criteria, and the addition of explicit utility specifications to allow for more principled evolution. Additionally, recent work in PSO suggests that connecting a Bayesian model of optimization to existing techniques may serve to clarify their intent and to strengthen their performance [Monson and Seppi 2005], something that should be pursued for EDAs and other evolutionary algorithms.

Chapter 9

Utile Function Optimization

To be Submitted to the Evolutionary Computation Journal

Abstract

The Evolutionary Optimization DBN (EO-DBN)—a dynamic Bayesian model of optimization—provides a unique opportunity to create an algorithm that more directly addresses the goal of optimization: to carefully select function samples so as to obtain information about the location of its global optimum. As thus described, optimization is fundamentally a decision process and will therefore be addressed using the language and tools of decision theory. Having once cast the problem as a probabilistic network (the EO-DBN), it is possible to create a decision-making agent that uses explicit definitions of utility and cost to rationally select sample locations that maximize information. This work presents and develops this idea, producing a model of optimization and a corresponding algorithm that is optimal with respect to well-stated optimization goals. The algorithm uses naturally expressed domain knowledge to determine where a function should be sampled and when the sampling process should stop, displaying sophisticated behavior when provided with simple specifications.

9.1 Introduction

Recent work in evolutionary computation has produced the Evolutionary Optimization DBN (EO-DBN), a Dynamic Bayesian Network (DBN) that expresses the information relationships present in all optimization problems. The model uses function samples to infer a distribution over likely locations for the global optimum, then generates new candidate locations by sampling from it [Monson and Seppi 2006]. In that sense it is similar in spirit to Estimation of Distribution Algorithms (EDAs) [Larrañaga et al. 1999; Larrañaga and Lozano 2001; Pelikan et al. 2002], but differs significantly in the way that the distribution is obtained: through a natural and explicit expression of the function class of interest.

That the function class is important is a necessary consequence of No Free Lunch (NFL) theorems for optimization, which state that any optimization algorithm that outperforms random search must do so on a limited subset or class of all possible functions [Macready and Wolpert 1996; Wolpert and Macready 1997; Whitley and Watson 2006; Igel and Toussaint 2004; Christensen and Oppacher 2001]. The EO-DBN gives expression to that class, taking a critical step towards a more complete understanding of the optimization problem and more principled approaches to the design of algorithms intended to solve it. For example, instead of developing behavioral heuristics to approach optimization, a researcher may produce a working algorithm by defining the function class; the EO-DBN then makes effective use of that information to infer the location of the optimum. In fact, because it relies on Bayesian inference, the distributions produced by the EO-DBN are optimal with respect to the function class specification and all acquired function samples [DeGroot 1970].

This inherited optimality is an important feature of the EO-DBN, but is conditioned upon the function samples actually taken; it says nothing about whether the samples themselves are usefully selected. In fact, although the EO-DBN's method of selecting new sample locations is sensible on the surface (exploring regions likely to contain the global optimum), there is no guarantee that its strategy adequately addresses the true goal of optimization: deciding where to sample so as to *provide information* about the location of the global optimum, not necessarily to *directly observe* it.

Optimization, therefore, is fundamentally a decision process, and a rational decision-making agent must operate in the presence of well-defined utilities and costs; without them, it is impossible to rank potential choices [DeGroot 1970]. Making effective use of such utilities to perform optimization is the focus of this work, which transforms the purely statistical EO-DBN into an decision-theoretic process with explicit utilities and costs: a Utile Function Optimizer (UFO).

The UFO shares its core model structure with the EO-DBN, a static Bayesian model called the Bayesian Optimization Network (BON). This critical component is reviewed first, and the problems with its placement in the EO-DBN are discussed. Utility is then introduced into the model, transforming it into a simple decision process. With utility defined, the obvious but often-abused decision methodology of Maximum Expected Utility (MEU) is explored and its connection to the EO-DBN established. In order to facilitate better use of exploratory samples, a brief tutorial of the Expected Value of Sample Information (EVSI) is given, after which it is used within the UFO, contrasting its effectiveness with the sub-optimal EO-DBN. Finally, a definition of sampling cost is introduced into the UFO, illustrating its flexibility and expressive power. This simple and straightforward addition serves as a catalyst for sophisticated algorithm behavior and provides the UFO with a natural stopping criterion.

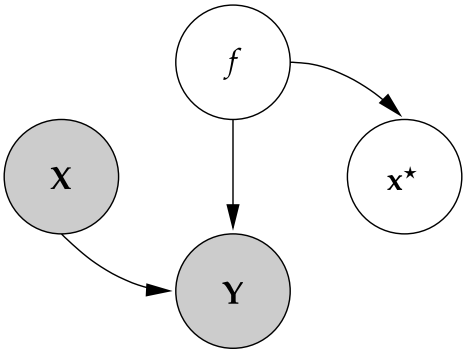


Figure 9.1: The Bayesian Optimization Network (shaded variables are observable)

9.2 The Bayesian Optimization Network

The BON, shown in Figure 9.1, is a graphical model that describes the relationships between variables that are present during the optimization process: a target function f, a vector of sample locations $\mathbf{X}$, and a corresponding vector of sample results $\mathbf{Y}$. As shown in the figure, f and $\mathbf{X}$ define or influence $\mathbf{Y}$, the individual elements of which may be scalar function outputs, gradient vectors, or any other information obtainable by querying f at each location in $\mathbf{X}$ (though this work will uniformly assume that only scalar values are obtained). All relationships are defined in terms of conditional probability distributions.

Because the goal of optimization is to find the location of the global optimum (generally assumed to be the maximum herein), that location is also a variable in the model, denoted $\mathbf{x}^\star$. This location is fixed and is one of the essential characteristics of the target function; the relationship between them is depicted as a link between f and $\mathbf{x}^\star$[1]. Shading in the figure is a mark of observability: $\mathbf{X}$ and $\mathbf{Y}$ are directly observable while f and $\mathbf{x}^\star$ are hidden. Note that even when a complete parametric definition of f is available, the node remains hidden because some of its essential

---

[1] While it is noteworthy that this link may describe a relationship between f and *any specified region of interest* (not merely the optimum), making use of that flexibility is more appropriately the subject of future work and will not be directly addressed here.

Algorithm 4 Particle filter for the BON

```
# Given a candidate population $\mathbf{X}$, sample the true function $f^\star$ to obtain results
$\mathbf{Y} = (\mathbf{y}_1 \dots \mathbf{y}_{N_\mathbf{X}})^\top$, with $\mathbf{y}_i = f^\star(\mathbf{x}_i)$ for $\mathbf{x}_i \in \mathbf{X}$
# Create a number of particles and calculate the normalized likelihood for each
$\mathbf{F} = (f_1 \dots f_{N_\mathbf{X}})^\top$, with $f_i \sim \rho_{f_t}(f_t)$
$\mathbf{\Lambda} = \left(\lambda_{f_1 \,|\, \mathbf{X},\mathbf{Y}} \dots \lambda_{f_{N_\mathbf{F}} \,|\, \mathbf{X},\mathbf{Y}}\right)^\top / \sum_{i=1}^{N_\mathbf{F}} \lambda_{f_i \,|\, \mathbf{X},\mathbf{Y}}$ with $\lambda_{f_i \,|\, \mathbf{X},\mathbf{Y}} = \prod_{j=1}^{N_\mathbf{X}} \rho_{\mathbf{y}_t \,|\, f_t, \mathbf{x}_t}\left(\mathbf{y}_j \,|\, f_i, \mathbf{x}_j\right)$
# The result is a discrete representation of the function posterior that may be sampled:
$\rho_{f \,|\, \mathbf{X},\mathbf{Y}}(f \,|\, \mathbf{X}, \mathbf{Y}) := \mathbf{F}, \mathbf{\Lambda}$
```

characteristics may be difficult or impossible to obtain from the definition, e.g., the location of $\mathbf{x}^\star$.

Optimization is accomplished by inferring the distribution $\rho_{\mathbf{x}^\star \,|\, \mathbf{X},\mathbf{Y}}$. Inasmuch as that distribution is accurate, its highest mode represents the location of $\mathbf{x}^\star$. The distribution is obtained by performing Bayesian inference in the BON:

$$\rho_{\mathbf{x}^\star \,|\, \mathbf{X},\mathbf{Y}}(\mathbf{x}^\star \,|\, \mathbf{X}, \mathbf{Y}) = \int \rho_{\mathbf{x}^\star \,|\, f}(\mathbf{x}^\star \,|\, f)\, \rho_{f \,|\, \mathbf{X},\mathbf{Y}}(f \,|\, \mathbf{X}, \mathbf{Y})\, df \tag{9.1}$$

where (by Bayes' Law)

$$\rho_{f \,|\, \mathbf{X},\mathbf{Y}}(f \,|\, \mathbf{X}, \mathbf{Y}) \propto \rho_{\mathbf{Y} \,|\, f,\mathbf{X}}(\mathbf{Y} \,|\, f, \mathbf{X})\, \rho_f(f) \quad . \tag{9.2}$$

As these distributions are not likely to be Gaussian or in any other way convenient, the application of a numerical inference method such as likelihood weighting is employed to obtain an approximation of $\rho_{\mathbf{x}^\star \,|\, \mathbf{X},\mathbf{Y}}$. Algorithm 4 details how this distribution is obtained in this and previous work [Monson and Seppi 2006]. Given a reasonable approximation, $\rho_{\mathbf{x}^\star \,|\, \mathbf{X},\mathbf{Y}}$ represents as much knowledge about $\mathbf{x}^\star$ as can be obtained from the observed data $\mathbf{X}, \mathbf{Y}$. In this sense, inference is optimal with respect to the amount of information gained about $\mathbf{x}^\star$ from the observations [DeGroot 1970].

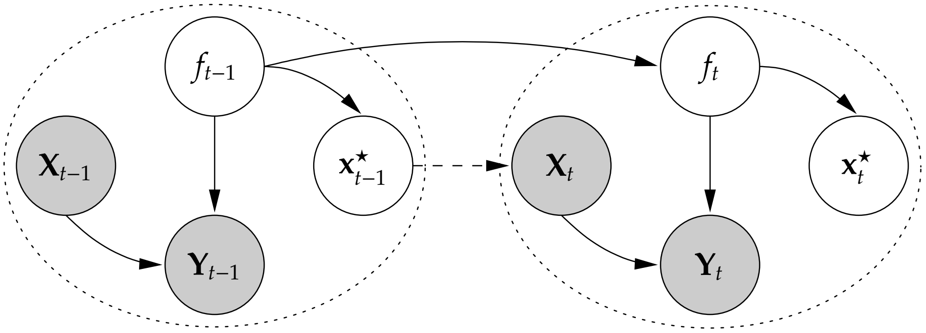


Figure 9.2: The Evolutionary Optimization DBN (EO-DBN)

Regardless of the applied method, (9.1), (9.2), and Algorithm 4 indicate that successful inference in the BON requires that the following distributions be specified:

- ρ_f: a *prior distribution* over all possible functions,
- $\rho_{\mathbf{Y}|f,\mathbf{X}}$: the *sampling distribution* describing how values are obtained, and
- $\rho_{\mathbf{x}^\star|f}$: the *goal distribution* relating functions in ρ_f to their global optima.

As detailed in previous work, the first two distributions define a function class. For example, a practitioner may be interested in optimizing any function that is "similar to a bowl". Such a subjective definition can be made precise by specifying the definition of "a bowl" in ρ_f and "similar to" in $\rho_{\mathbf{Y}|f,\mathbf{X}}$. The third distribution $\rho_{\mathbf{x}^\star|f}$ describes the intent of search, be it finding an optimum or some other area of interest; it simply defines the relationship between the *supplied definition* of "a bowl" (not the target function itself) and an important region in that bowl. This approach to defining the core elements of optimization is straightforward, precise, and clear [Monson and Seppi 2006].

Having obtained information from one population of samples, it is often desirable to select a new population, using the information obtained to refine $\rho_{\mathbf{x}^\star|\mathbf{X},\mathbf{Y}}$; this gives the model an essentially evolutionary characteristic. The EO-DBN, which consists of several BONs replicated over time, is one such evolutionary model and is depicted in Figure 9.2. The horizontal link between function variables can be used to specify a function with dynamic properties through the use of an additional distribution $\rho_{f_t \mid f_{t-1}}$, generating a model that is strictly more expressive than the BON.

In contrast, the link between $\mathbf{x}^\star_{t-1}$ and $\mathbf{X}_t$ adds no expressive power to the model, but defines a population creation strategy where new candidates are sampled from $\rho_{\mathbf{x}^\star \mid \mathbf{X},\mathbf{Y}}$. This distribution acts as a simple decision-making agent, selecting new sample points based on the likelihood that they produce optimal function output. While all other variables and relationships in the EO-DBN are clearly motivated and explicitly defined, this particular relationship is much less so; on the surface it seems sensible to explore the target domain in regions that are likely to contain the global maximum, but there is no guarantee that the values at these locations will provide useful information.

Decision-making agents can only act rationally in the presence of well-defined utilities, making it impossible to support or refute any claims about the strategy employed by the EO-DBN without first isolating those utilities on which its decisions are based. In this instance, it assigns higher utility to sample locations that are likely to be statistically near the global optimum, an approach that is reasonable if the goal of optimization is to produce high output with every possible sample.

Unfortunately, this last statement is nonsense. The goal of *exploitation* is to produce high output with every sample. The goal of optimization, on the other hand, is to *discover* sample locations that produce maximal output; if the global optimum is never sampled, but its location becomes clear, then optimization is successful. Optimization is essentially a directed information-gathering process, a process described by other communities as *active sampling*, *active learning*, *active selection*, or *experimental design* [Blum and Langley 1997; Fedorov 1972].

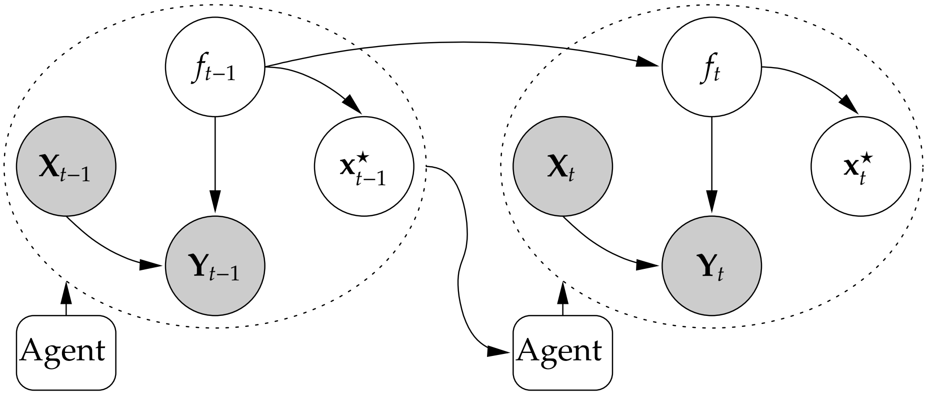


Figure 9.3: Utile Function Optimizer (UFO)

This work addresses the question of how to assign and rationally employ utilities so that an agent can achieve the fundamental goal of empirical optimization: discovering the optimum as quickly and accurately as possible through judicious sampling. In other words, it provides instructions on how to achieve an "optimal allocation of trials" for continuous optimization [Holland 1973]. It does so by first defining a simple and easily motivated utility function for use in conjunction with the BON.

9.3 The Utile Function Optimizer

Because it is senseless to introduce utility into a model that leaves no room for decisions, it is necessary to replace the link between $\mathbf{x}^{\star}_{t-1}$ and $\mathbf{X}_t$ with a decision-making agent that can make use of utility. This basic transformation results in the Utile Function Optimizer (UFO) depicted in Figure 9.3[2]. The agent nodes in the figure can represent any decision-maker, including existing evolutionary algorithms or a human. This work will primarily be concerned with creating an

---

[2] The network has a structure similar to a Partially Observable Markov Decision Process (POMDP), but is fundamentally continuous, has non-stationary transitions, and is utilized quite differently than a traditional POMDP. Therefore, while noting the similarities, this work will not generally refer to the network as a POMDP.

agent that behaves rationally, i.e., an agent that makes its decisions based on the maximization of utility.

Utility functions can be rather arbitrary, ranging anywhere from simple and uncontroversial to complex harbingers of contention. The following utility function favors simplicity and generality:

$$u_{f_t,\mathbf{x}_t}(f_t, \mathbf{x}_t) = f_t(\mathbf{x}_t) \ . \tag{9.3}$$

Simply put, this function assigns greater value to locations that produce larger outputs in the function; maximizing it will also maximize the function, achieving optimization. Although on the surface this definition is pedestrian and even redundant, it has been carefully defined and is sufficient to produce surprisingly sophisticated behavior when combined with the information inferred from the UFO model.

This simple utility definition will be used as the basis for decisions throughout the rest of this work. Its first use will be direct, maximizing its expected value. This greedy approach is intended to illustrate the underlying meaning of the EO-DBN's strategy and the reasons that it is a poor choice as a means of accomplishing the fundamentally exploratory goal of optimization.

9.4 Maximum Expected Utility

Given a queryable function f, the utility function in (9.3) provides an approach to determining the intrinsic value of a sample location $\mathbf{x}$. Therefore, if f is known, it can be used to make a rational decision, e.g., one that selects the $\mathbf{x}$ that maximizes utility. In practice, however, this is impossible because the true value of the variable f is hidden. The BON provides a way around this issue because it can be used to obtain a distribution over f (either a prior or a posterior). With that distribution it is possible to compute the *expected value* of the utility for a particular location $\mathbf{x}$, and that expectation can be used by a rational agent.

We write the expected value of a function using the notation of expectation, where any variable *not* appearing in the subscript is integrated away, e.g.,

$$\mathrm{E}_{y\,|\,x}\left[g(y,z)\right] = \int g(y,z)\rho_{Z\,|\,X}(z\,|\,x)\;dz \;\; .$$

It should be noted that at least two popular and diametrically opposed notations are used in practice, and that we have selected one that is consistent with the notations for probability and utility employed in this work. The reader who has prior experience with the opposing notation (where subscripts *are* integrated out) should therefore take care to perform the necessary mental reversals when examining the formulas that follow.

The agent described in this section is denoted MEU because it maximizes the Expected Utility (EU). If exploitation is desired (sampling so that high function output is actually observed and not merely inferred), then this is rational agent behavior; MEU produces a fundamentally exploitative agent. In order to make decisions, this agent must therefore compute the location that maximizes expected utility:

$$\mathrm{E}_{\mathbf{x}_t\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1}}\left[u_{f_t,\mathbf{x}_t}(f_t,\mathbf{x}_t)\right] = \int u_{f_t,\mathbf{x}_t}(f_t,\mathbf{x}_t)\,\rho_{f_t\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1}}(f_t\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1})\;df_t \;\; . \tag{9.4}$$

The details of this approach and its impact on the optimization process are illustrated in the following example.

9.4.1 MEU Example: Cone Class

This example uses a maximizing version of the intentionally simplistic function class "similar to a cone" [Monson and Seppi 2006]:

$$\rho_f(f) = \begin{cases} \rho_{\mathbf{c}}(\mathbf{c}) & \text{if } f \in \{\lambda \mathbf{x}.g(\mathbf{x},\mathbf{c}) \mid \forall \mathbf{x} \in \mathbb{R}^D.\ g(\mathbf{x},\mathbf{c}) = -\left\|\mathbf{x}-\mathbf{c}\right\|_2\} \\ 0 & \text{otherwise} \end{cases} \tag{9.5}$$

$$\rho_{\mathbf{c}}(\mathbf{c}) = N(\mathbf{0}, \Sigma_{\mathbf{c}}) \tag{9.6}$$

$$\rho_{\mathbf{y}\,|\,f,\mathbf{x}}(\mathbf{y} \mid f, \mathbf{x}) = N(f(\mathbf{x}), \sigma_{\mathbf{y}}) \tag{9.7}$$

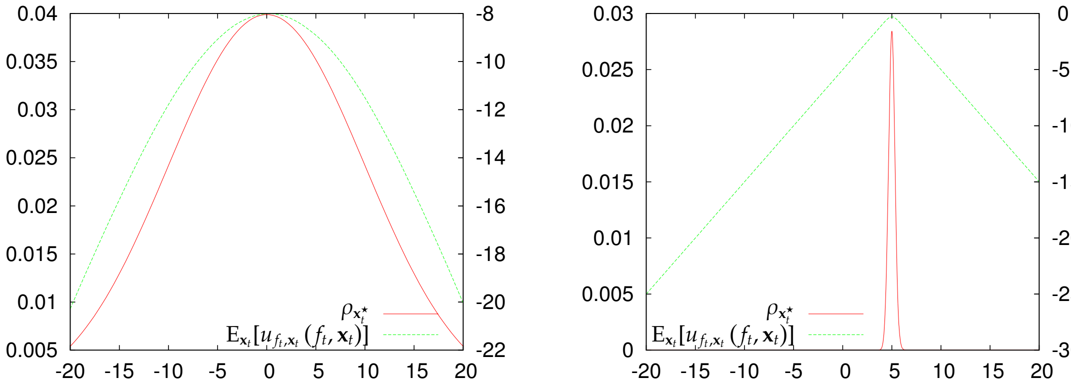
$$\rho_{\mathbf{x}^\star\,|\,f}(\mathbf{x}^\star \mid f) = \delta(\mathbf{x}^\star - \mathbf{c}) \ . \tag{9.8}$$

The prior defined in (9.5) and (9.6) indicates that the desired function class contains only symmetric cones of a particular kind, favoring those whose centers are obtainable by sampling a Normal distribution about the origin. The sampling distribution in (9.7) admits functions whose values are within a Gaussian draw of the value of a cone, and the goal distribution in (9.8) points to the maximum of a given cone. The omission of time subscripts indicates that these distributions are the same for all values of t.

Let the true function be part of the described class: a noise-free, one-dimensional cone centered at $\mathbf{x} = 5$, with additional class parameters specified as follows:

$$f^\star(\mathbf{x}) = -\left\|\mathbf{x}-5\right\|_2 \qquad \Sigma_{\mathbf{c}} = 10^2\mathbf{I} \qquad \sigma_{\mathbf{y}} = 1 \ . \tag{9.9}$$

At $t = 0$, the distribution ρ_{f_t} required by (9.4) is simply the prior from (9.5) and (9.6). The corresponding distribution over maxima ($\rho_{\mathbf{x}_t^\star}$) and the expected utility curve ($\mathrm{E}_{\mathbf{x}_t}[u_{f_t,\mathbf{x}_t}(f_t, \mathbf{x}_t)]$) are shown together in Figure 9.4(a). Similar graphs at $t = 1$ are shown in Figure 9.4(b), obtained after observing $\mathbf{X}_0$ and $\mathbf{Y}_0$.



(a) $t = 0$

(b) $t = 1$

Figure 9.4: Distribution over $\mathbf{x}^\star$ (left axis) and expected utility (right axis)

The graphs highlight some important concepts. First, the introduction of data represents information that changes the distribution over possible maxima: the first is due entirely to the prior and is broad and wrongly positioned; the second is narrow and centered on the correct optimum. Second, in both cases the peak of $\rho_{\mathbf{x}^\star}$ coincides directly with the peak of the expected utility function.

This last point is potentially significant for the EO-DBN and other existing evolutionary algorithms: these algorithms employ some amount of greedy selection, e.g., the EO-DBN selects points that are already most likely to be correct, PSO particles oscillate around attractors that represent the best locations seen so far, Genetic Algorithms (GAs) prune their populations using fitness-based selection before employing recombination, EDAs use the fittest members for distribution estimation, simulated annealing only selects worse locations with decreasing probability, and amoeba search often relocates its worst point while leaving the others stationary. In short, the idea of pursuing samples based on greedy heuristics is pervasive in evolutionary optimization. That MEU is similarly greedy is no surprise; it is, after all, maximizing its expected utility and was known to be exploitative before being applied. Because it assumes a function class that is monotonic away from the optimum, a trait shared with the posterior $\rho_{\mathbf{x}^\star \mid \mathbf{X},\mathbf{Y}}(\mathbf{x}^\star \mid \mathbf{X}, \mathbf{Y})$, it is also not surprising that it selects precisely the same sample points as the EO-DBN.

This may shed light on some of the hidden assumptions in existing evolutionary algorithms, especially those that make use of an explicit notion of fitness (utility). These approaches assume not only that better values have more intrinsic worth when sampled (fitness maximization), but that those values will tend to be close to one another, indicating that the function class of interest is likely to be shaped, for lack of a better description, like a cone. When this assumption is violated, it is only natural that the algorithms fail, giving rise to research on the significance and difficulty of "deceptive functions", functions with gradients that often point away from the goal. That "deception" has long been considered to be a leading cause in the failure of evolutionary algorithms (subject, of course, to debate [Grefenstette 1993; Horn 1985]) is supported by these results.

The connection between expected utility in (9.3) and $\rho_{\mathbf{x}_t \mid \mathbf{x}^\star_{t-1}}$ is now clear: provided a function class that is not "deceptive" and a straightforward fitness-based utility, sampling from $\rho_{\mathbf{x}_t \mid \mathbf{x}^\star_{t-1}}$ is more or less the same as selecting locations via MEU. Provided that attempting to obtain high-fitness samples from the function is the goal of the agent, this is perfectly rational agent behavior. As discussed previously, however, the goal of optimization is not necessarily to *produce* good values; that is the goal of exploitation once the location of the optimum is known. Instead, the goal of optimization is to *discover* the location that will produce maximal utility so that it may be obtained when exploitation is performed in the future.

While MEU fails to accomplish that goal, the tools to do so are now available; armed with an explicit definition of exploitative utility, the inherently greedy MEU can be replaced with something that favors information over value: The Expected Value of Sample Information. A brief tutorial of this useful concept follows.

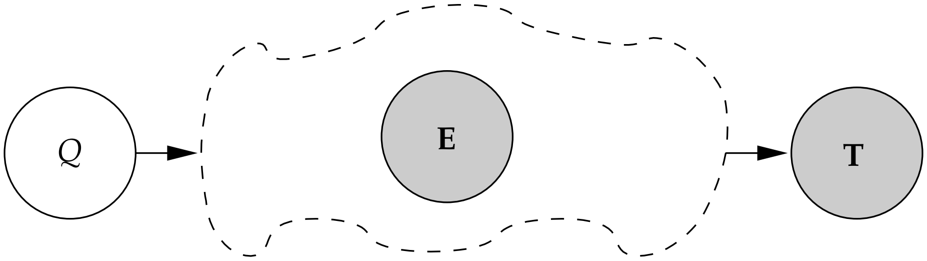


Figure 9.5: A general Bayesian network suitable for EVSI

9.5 A Brief EVSI Tutorial

There are many ways of using utility to ensure that maximization of some utility-related quantity corresponds to maximization of information gain. Perhaps the most straightforward approach is to directly redefine $u_{f_t,\mathbf{x}_t}$ such that higher value is assigned to points with higher information content. While useful, this particular approach requires the definition and application of an additional *information measure*, typically related to Shannon Entropy [Fedorov 1972; MacKay 1992a,b]. There is a way of accomplishing the same end, however, without introducing any new measures or inventing special-purpose non-exploitative utility functions: by calculating the Expected Value of Sample Information (EVSI)[3] [Lindley 1985].

The environment in which EVSI operates is essentially a Bayesian network like that shown in Figure 9.5. The network may consist of arbitrary connections and nodes provided that some minimal concepts are represented: a requisite *query variable* Q, an optional set of *evidence variables* **E**, and *test variables* **T**. Note that whatever the position of these variables in the original network, Bayesian reasoning always admits transformations such that the general structure is that shown.

The query variable Q represents information that is desired but hidden, requiring inference to obtain a distribution over its possible values. The evidence variables **E** are observable and the values of the test variables **T** represent results

---

[3] EVSI is sometimes referred to as the Expected Value of Partial Information, as is the case in this reference.

that are obtainable but not yet known. These results may be selectively acquired (or *sampled*) by performing one or more of the tests.

The variables in the network represent information used in a decision process; the purpose of evaluating Q is to determine the appropriate value for a decision variable D. In order to accomplish this, a utility must be available that is defined on these two variables $u_{D,Q}$. Given the conditional and prior probabilities that make up the network's definition, the utility function $u_{D,Q}$, evidence $\mathbf{E}$, and tests $\mathbf{T}$, it is possible to calculate the expected value of performing a particular test *Test*:

$$EVSI_{Test\,|\,\mathbf{e}} = \mathrm{E}_{|\,\mathbf{e}}\left[\max_{d} \mathrm{E}_{d\,|\,\mathbf{e},t}\left[u_{Q,D}\left(q,d\right)\right]\right] - \max_{d} \mathrm{E}_{d\,|\,\mathbf{e}}\left[u_{Q,D}\left(q,d\right)\right] \tag{9.10}$$

where

$$\mathrm{E}_{d\,|\,\mathbf{e}}\left[u_{Q,D}\left(q,d\right)\right] = \int u_{Q,D}\left(q,d\right) \rho_{Q\,|\,\mathbf{E}}\left(q\,|\,\mathbf{e}\right)\; dq \tag{9.11}$$

$$\mathrm{E}_{d\,|\,\mathbf{e},t}\left[u_{Q,D}\left(q,d\right)\right] = \int u_{Q,D}\left(q,d\right) \rho_{Q\,|\,\mathbf{E},Test}\left(q\,|\,\mathbf{e},t\right)\; dq\ . \tag{9.12}$$

The first term of (9.10) is an expectation over the MEU for all test outcomes and the second is the MEU in the absence of test information.

The idea behind EVSI is that the maximum expected utility increases in the presence of information; EVSI is simply a principled way of quantifying the increase. Consider for a moment what this means: given the opportunity to gather information, EVSI indicates what that information is likely to be worth in the units of specified utility. When multiple kinds of information can be gathered, EVSI can be used to determine what information will yield the greatest potential benefit. Additionally, if the information has some associated cost, this method indicates whether new information should be gathered at all: if the expected improvement in utility does not exceed the cost of sampling, then sampling should not be done. This concept is perhaps best understood by way of a simple example.

9.5.1 EVSI Example: Drilling For Oil

Consider an oil drilling company that has an opportunity to purchase and drill a plot of land for \$1 million. The site may or may not contain oil, but if it does, the oil is worth about \$10 million. Given this information, a utility function may be defined in terms of the query variable *Oil* and the decision variable *Drill*:

$$u_{Drill,Oil}(\mathrm{T},\mathrm{T}) = \$9\text{ million} \qquad u_{Drill,Oil}(\mathrm{T},\mathrm{F}) = -\$1\text{ million} \tag{9.13}$$

$$u_{Drill,Oil}(\mathrm{F},\mathrm{T}) = \$0 \qquad u_{Drill,Oil}(\mathrm{F},\mathrm{F}) = \$0\ . \tag{9.14}$$

If the site's potential for containing oil is known in terms of the prior probabilities, e.g., $Pr_{Oil}(\mathrm{T}) = 0.6$, then the most rational action can be calculated using the expected utility of drilling ($d = \mathrm{T}$) and not drilling ($d = \mathrm{F}$):

$$\mathrm{E}_d\left[u_{Drill,Oil}(d,o)\right] = \sum_{o\in\{\mathrm{T},\mathrm{F}\}} u_{Drill,Oil}(d,o)\, Pr_{Oil}(o) = \begin{cases} \$5\text{ million} & \text{if } d = \mathrm{T} \\ \$0 & \text{if } d = \mathrm{F} \end{cases}\ . \tag{9.15}$$

On average, drilling a plot like this one will produce a profit of \$5 million, so the correct choice is to purchase the plot. This result, however, does not say much about whether this *specific* plot of land should be drilled; it only indicates that if a large set of similar plots is available, then purchasing and drilling them is a good idea *on average*. Given a single plot, it is unlikely that a company would purchase it based on such a result: it would instead perform some kind of test before making a purchase decision.

Having more information will clearly improve the company's chances of making a fruitful purchase decision, so the availability of the test transforms the problem from one of deciding whether to purchase the plot into one of whether to incur the cost of a test before making a decision; this is what EVSI computes. Suppose, then, that the test costs \$0.1 million and has the following accuracy:

$$Pr_{Test\,|\,Oil}(\mathrm{T}\,|\,\mathrm{T}) = 0.95 \qquad Pr_{Test\,|\,Oil}(\mathrm{T}\,|\,\mathrm{F}) = 0.20\ . \tag{9.16}$$

The test is not as accurate as may be desired, but nonetheless provides useful information. Because the outcome of the test may change the company's decision, the new expected utility of drilling is conditioned upon it:

$$\mathrm{E}_{d\,|\,t}\left[u_{Drill,Oil}(d,o)\right] = \sum_{o\in\{\mathrm{T},\mathrm{F}\}} u_{Drill,Oil}(d,o)\, Pr_{Oil\,|\,Test}(o\,|\,t) \tag{9.17}$$

where $Pr_{Oil\,|\,Test}(o\,|\,t)$ is the Bayesian posterior

$$Pr_{Oil\,|\,Test}(o\,|\,t) = \frac{Pr_{Test\,|\,Oil}(t\,|\,o)\, Pr_{Oil}(o)}{Pr_{Test}(t)} \tag{9.18}$$

$$Pr_{Test}(t) = \sum_{o\in\{\mathrm{T},\mathrm{F}\}} Pr_{Test\,|\,Oil}(t\,|\,o)\, Pr_{Oil}(o) = 0.65 \tag{9.19}$$

giving that

$$Pr_{Oil\,|\,Test}(\mathrm{T}\,|\,\mathrm{T}) \approx 0.88 \qquad Pr_{Oil\,|\,Test}(\mathrm{F}\,|\,\mathrm{T}) \approx 0.12 \tag{9.20}$$

$$Pr_{Oil\,|\,Test}(\mathrm{T}\,|\,\mathrm{F}) \approx 0.09 \qquad Pr_{Oil\,|\,Test}(\mathrm{F}\,|\,\mathrm{F}) \approx 0.91 \tag{9.21}$$

and finally

$$\mathrm{E}_{d\,|\,t}\left[u_{Drill,Oil}(d,o)\right] = \begin{cases} \$7.8\text{ million} & \text{if } t = \mathrm{T}, d = \mathrm{T} \\ -\$0.1\text{ million} & \text{if } t = \mathrm{F}, d = \mathrm{T} \\ \$0 & \text{if } d = \mathrm{F} \end{cases} \ . \tag{9.22}$$

As expected, more information increases the expected utility of drilling. If the test outcome is negative, it also changes the decision; drilling should be avoided.

Having knowledge of the test's outcome is desirable; it guides the decision-making process such that maximum expected utility can be increased. The test itself costs \$0.1 million, however, so it is useful to know what the *expected improvement* in maximum utility will be given the test's outcome *before it is performed*. EVSI is the right way to calculate this while taking the accuracy of the test into account:

$$\begin{aligned}
EVSI_{Test} &= \mathrm{E}\left[\max_{d} \mathrm{E}_{d\,|\,t}\left[u_{Oil,Drill}\left(o,d\right)\right]\right] - \max_{d} \mathrm{E}_{d}\left[u_{Oil,Drill}\left(o,d\right)\right] && (9.23)\\
&= \mathrm{E}\left[\begin{cases} \$7.8 \text{ million} & \text{if } t = \mathrm{T} \\ \$0 & \text{if } t = \mathrm{F} \end{cases}\right] - \$5 \text{ million} && (9.24)\\
&= \left(\left(\$7.8 \text{ million}\right) Pr_{Test}\left(\mathrm{T}\right) + \left(\$0\right) Pr_{Test}\left(\mathrm{F}\right)\right) - \$5 \text{ million} && (9.25)\\
&= \$0.07 \text{ million} && (9.26)
\end{aligned}$$

Obtaining test results is therefore worth about \$0.07 million on average, but the test is not cost-effective because it costs \$0.1 million. Were the test more accurate, the expected value improvement would increase and it would perhaps be worth its price; as it is, a lower price should be negotiated or a more accurate test should be performed.

9.6 EVSI in the UFO

EVSI is both powerful and simple, answering the question of whether new information is expected to be worth its cost. In the process it also answers a question that is of supreme importance in function optimization: "How will immediate exploration affect future exploitation?" Optimization is fundamentally concerned with finding a location that, *if exploited*, will produce a good value, not necessarily with sampling that value immediately. The application of EVSI to optimization is a principled and meaningful way to accomplish that goal: active sampling for maximization of information about the global minimum.

The application of EVSI to the UFO can answer the fundamental question of empirical optimization: "What sample locations will provide the most information about the location of the global optimum?" Unlike the oil drilling example, however, it is easy to confound tests with decisions when performing optimization, requiring careful development of the approach in this context.

In the previous example, the decision is to drill or not to drill. The test, on the other hand, is separate from that decision (e.g., a geological survey) and has its own associated costs. It is conceivable, however, that the test could actually consist of drilling the land; after all, that would be an excellent (but costly) indicator of the presence or absence of oil. Empirical function optimization is similar to that situation; an infinity of tests are available in the form of potential sample locations, and the outcome of those tests will affect knowledge of $\mathbf{x}^\star$: the place that would be sampled during exploitation (MEU).

In other words, the distinction between tests and decisions is the same as the distinction between exploration and exploitation: the test is performed in the hopes of obtaining more information, and the decision is made in order to obtain actual value. EVSI indicates the value of an exploratory test, and EU computes the value of an exploitative decision. That they operate in the same domain is perfectly acceptable, but care must be taken with the notation.

When using the UFO, a test variable and its outcome will be distinguished by a superscript $^?$: the candidate population $\mathbf{X}_t^?$ represents one possible test, and its outcome is denoted $\mathbf{Y}_t^?$. The decision, on the other hand, is also a population of locations $\mathbf{X}_t$, locations that would be given high value by EU. The test is *concrete*, representing real samples, and the decision is *hypothetical* in the calculation of

EVSI (as evidenced by the fact that decisions only appear in the context of max operator). The query variable, representing desired information, must be one of the utility function parameters and is in this case f_t, from which the truly important information $\mathbf{x}_t^\star$ may be obtained.

Assuming a candidate population of size 1, EVSI in the presence of the UFO model is given to be

$$EVSI_{\mathbf{x}_t^? \,|\, \mathbf{e}_t} = \mathrm{E}_{\,|\, \mathbf{e}_t, \mathbf{x}_t^?} \left[\max_{\mathbf{x}_t} \mathrm{E}_{\mathbf{x}_t \,|\, \mathbf{e}_t, \mathbf{x}_t^?, \mathbf{y}_t^?} \left[u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \right] \right] - \max_{\mathbf{x}_t} \mathrm{E}_{\mathbf{x}_t \,|\, \mathbf{e}_t} \left[u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \right] \quad . \tag{9.27}$$

Unlike the oil drilling example, here EVSI makes use of evidence $\mathbf{e}_t = (\mathbf{X}_{t-1}, \mathbf{Y}_{t-1})$. Splitting the calculations so that the required distributions are evident yields the following:

$$\mathrm{E}_{\mathbf{x}_t \,|\, \mathbf{e}_t, \mathbf{x}_t^?, \mathbf{y}_t^?} \left[u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \right] = \int u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \rho_{f_t \,|\, \mathbf{e}_t, \mathbf{x}_t^?, \mathbf{y}_t^?} \left(f_t \,|\, \mathbf{e}_t, \mathbf{x}_t^?, \mathbf{y}_t^? \right) \, df_t \tag{9.28}$$

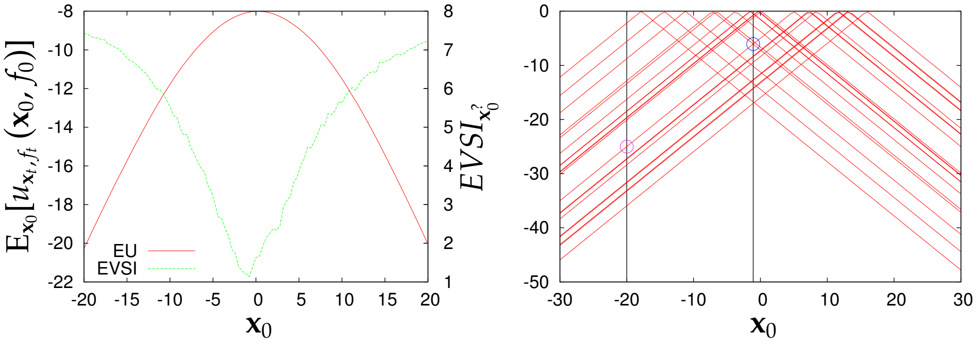
$$\mathrm{E}_{\mathbf{x}_t \,|\, \mathbf{e}_t} \left[u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \right] = \int u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t, f_t \right) \rho_{f_t \,|\, \mathbf{e}_t} \left(f_t \,|\, \mathbf{e}_t \right) \, df_t \quad . \tag{9.29}$$

Additionally, computation of the outer expectation in the first term of (9.27) requires the distribution $\rho_{\mathbf{y}_t^? \,|\, \mathbf{X}_{t-1}, \mathbf{Y}_{t-1}, \mathbf{x}_t^?}$. All necessary distributions may be obtained from the network using empirical methods such as particle filters, the method of choice for this work. The computation of the posterior $\rho_{f|\mathbf{X},\mathbf{Y}}$ is performed as before with Algorithm 4, and the method for computing EVSI in this context is supplied as Algorithm 5.

Applying this algorithm provides the optimization practitioner with important information: where next to sample $f^\star$ so that maximal information may be obtained about $\mathbf{x}^\star$. Specific examples that demonstrate this predictable behavior are given next.

Algorithm 5 $EVSI_{\mathbf{x}^?\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1}}$ in the UFO using a particle filter

```
# Create several empirical distributions as bags of values (chain rule)
$\mathbf{F} = (f_1 \ldots f_{N_\mathbf{F}})^\top$, where $f_i \sim \rho_{f_t\,|\,\mathbf{e}_t}(\cdot\,|\,\mathbf{X}_{t-1}, \mathbf{Y}_{t-1})$
$\mathbf{X} = (\mathbf{x}_1 \ldots \mathbf{x}_{N_\mathbf{F}})$, where $\mathbf{x}_i \sim \rho_{\mathbf{x}^\star_t\,|\,f_t}(\cdot\,|\,f_i)$ for $f_i \in \mathbf{F}$
$\mathbf{Y}^? = \left(\mathbf{y}^?_1 \ldots \mathbf{y}^?_{N_\mathbf{F}}\right)$, where $\mathbf{y}^?_i \sim \rho_{\mathbf{y}_t\,|\,f_t,\mathbf{x}_t}\left(\cdot\,|\,f_i, \mathbf{x}^?\right)$ for $f_i \in \mathbf{F}$
# Calculate EVSI
$T = \frac{1}{N_\mathbf{F}} \sum_{\mathbf{y}^? \in \mathbf{Y}^?} \max_{\mathbf{x} \in \mathbf{X}} \sum_{f \in \mathbf{F}} u_{\mathbf{x},f}(\mathbf{x}, f)\, \rho_{\mathbf{y}_t\,|\,f_t,\mathbf{x}_t}\left(\mathbf{y}^?\,|\,f, \mathbf{x}^?\right)$
$M = \max_{\mathbf{x} \in \mathbf{X}} \sum_{f \in \mathbf{F}} u_{\mathbf{x},f}(\mathbf{x}, f)\, \rho_{f_t\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1}}(f\,|\,\mathbf{X}_{t-1}, \mathbf{Y}_{t-1})$
$EVSI_{\mathbf{x}^?\,|\,\mathbf{X}_{t-1},\mathbf{Y}_{t-1}} = T - M$
```



(a) EU and EVSI at $t = 0$

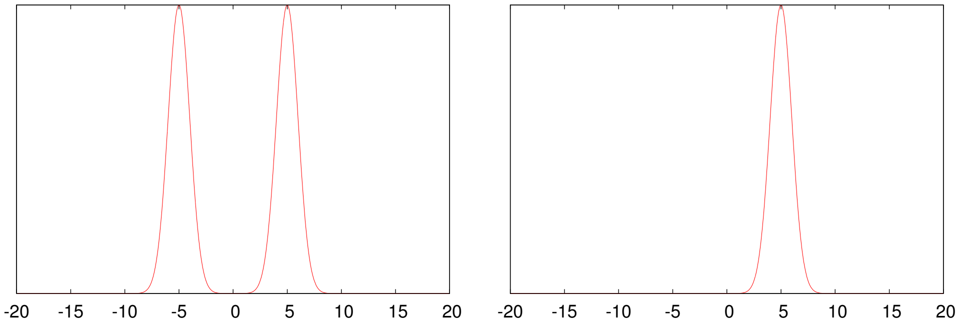
(b) Cones sampled from ρ_{f_0}, two points

Figure 9.6: EU vs. EVSI for the cone class

9.6.1 EVSI Example: Cone Class

Consider again the conic function class as described in (9.5)–(9.8) and the one-dimensional true function $f^\star(\mathbf{x}_t) = \|\mathbf{x}_t - 5\|_2$. Even in the absence of evidence, EVSI may be applied to the network to determine where $f^\star$ should first be sampled to provide maximal information. The results of that application are shown in Figure 9.6(a). Note that the prior $\rho_{\mathbf{x}_0^\star}$ and EU alike indicate that the origin is a likely candidate for $\mathbf{x}^\star$ but that $EVSI_{\mathbf{x}_0^?}$ suggests precisely the opposite: points near the origin are the least informative.

This result is instructive because, while initially counterintuitive, it is also perfectly reasonable. Points sampled near (but not at) the origin have values which are consistent with basically two cones in a function class that favors the



(a) Point selected according to MEU (b) Point selected according to EVSI

Figure 9.7: MEU vs. EVSI: affect of sample points on the posterior $\rho_{\mathbf{x}_0^\star|\mathbf{X}_0,\mathbf{Y}_0}$

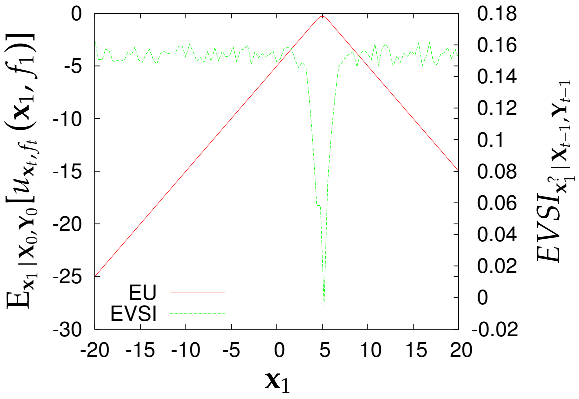


Figure 9.8: EU and EVSI at $t = 1$

origin, whereas points further away tend to be consistent with only one cone as illustrated in Figure 9.6(b). EVSI is correct in suggesting that distant points have more discriminating power.

That points selected according to MEU provide less information than those indicated by maximizing EVSI is illustrated forcefully in Figure 9.7, which shows the difference between resulting posteriors when selecting maximum value samples according to EU and EVSI. When sampling near the origin as MEU dictates, the posterior is that shown in Figure 9.7(a), which has two peaks as predicted. When sampling away from the origin as dictated by maximizing EVSI, the posterior distribution shown in Figure 9.7(b) has only one peak, again as predicted. EVSI has simply indicated more informative sample points than EU.

Figure 9.8 shows the difference between EU and EVSI at $t = 1$ when some evidence is present. The same reasoning applies to this graph as applies to the first: sampling away from locations of maximum expected utility (and also $\rho_{\mathbf{x}_0^\star | \mathbf{X}_0, \mathbf{Y}_0}$ in this case) provides more information about $\mathbf{x}^\star$ than otherwise.

The point of this example is to demonstrate that EVSI suggests rational sample locations that provide needed information for optimization. While results of its application are initially counterintuitive, upon closer inspection they are reasonable and correct. Because it produces simple and predictable results, it is tempting to develop a heuristic, apply it to an existing algorithm, and declare victory. This approach may work well on the cone class, but as soon as the function class changes it will fail. Part of the contribution of the UFO is the ability to apply EVSI to the optimization *regardless of the specific function class*; the algorithm is fixed, automatically adapting to the specified intent of the practitioner. Without the perspective on optimization provided by the UFO, this would simply not be possible.

To emphasize the fact that a different function class yields different results, another example follows.

9.6.2 EVSI Example: Bowl Class

Application to the more general "bowl-shaped" class [Monson and Seppi 2006] serves to shed more light on EVSI's behavior. That class, adapted for maximization and for use with EVSI, is reproduced here (showing only those distributions that

Algorithm 6 $EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^? \mid \mathbf{X}_{t-1},\mathbf{Y}_{t-1}}$ in the UFO for the "bowl" class

```
# Create several empirical distributions as bags of values (chain rule)
$\mathbf{F} = (f_1 \dots f_{N_\mathbf{F}})^\top$, where $f_i \sim \rho_{f_t \mid \mathbf{X}_{t-1},\mathbf{Y}_{t-1}}(\cdot \mid \mathbf{X}_{t-1}, \mathbf{Y}_{t-1})$
$\mathbf{X} = (\mathbf{x}_1 \dots \mathbf{x}_{N_\mathbf{F}})$, where $\mathbf{x}_i \sim \rho_{\mathbf{x}_t^\star \mid f_t}(\cdot \mid f_i)$ for $f_i \in \mathbf{F}_t$
$\mathbf{r}_\mathbf{y}^? = \left(r_1^? \dots r_{N_\mathbf{F}}^?\right)$, where $r_i^? \sim Pr_{r_\mathbf{y} \mid r_f}\left(\cdot \mid \text{sign}\left(f_i(\mathbf{x}_1^?) - f_i(\mathbf{x}_2^?)\right)\right)$ for $f_i \in \mathbf{F}_t$
# Calculate EVSI
$T = \frac{1}{N_\mathbf{F}} \sum_{r^? \in \mathbf{r}_\mathbf{y}^?} \max_{\mathbf{x}\in\mathbf{X}} \sum_{f\in\mathbf{F}} u_{\mathbf{x},f}(\mathbf{x}, f)\, Pr_{r_\mathbf{y}\mid r_f}\left(r^? \mid \text{sign}\left(f(\mathbf{x}_1^?) - f(\mathbf{x}_2^?)\right)\right)$
$M = \max_{\mathbf{x}\in\mathbf{X}} \sum_{f\in\mathbf{F}} u_{\mathbf{x},f}(\mathbf{x}, f)\, \rho_{f_t\mid\mathbf{X}_{t-1},\mathbf{Y}_{t-1}}(f \mid \mathbf{X}_{t-1}, \mathbf{Y}_{t-1})$
$EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^? \mid \mathbf{X}_{t-1},\mathbf{Y}_{t-1}} = T - M$
```

differ from the cone class):

$$\rho_{\mathbf{Y}_t \mid f_t, \mathbf{X}_t}(\mathbf{Y}_t \mid f_t, \mathbf{X}_t) = \rho_{\mathbf{r}_{\mathbf{Y}_t} \mid \mathbf{r}_{f_t,\mathbf{X}_t}}\left(\mathbf{r}_{\mathbf{Y}_t} \mid \mathbf{r}_{f_t,\mathbf{X}_t}\right) \tag{9.30}$$

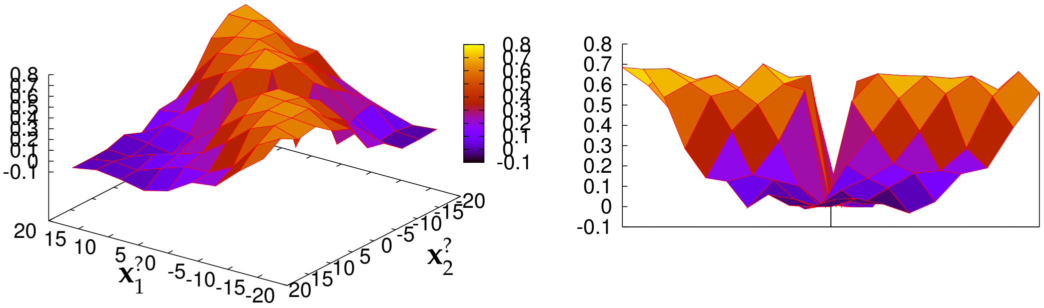
$$\rho_{\mathbf{r}_{\mathbf{Y}_t} \mid \mathbf{r}_{f_t,\mathbf{X}_t}}\left(\mathbf{r}_{\mathbf{Y}_t} \mid \mathbf{r}_{f_t,\mathbf{X}_t}\right) = \prod_{i<j} Pr_{r_\mathbf{y} \mid r_f}\left(\text{sign}\left(\mathbf{y}_i - \mathbf{y}_j\right) \mid \text{sign}\left(f_t(\mathbf{x}_i) - f_t(\mathbf{x}_j)\right)\right) \tag{9.31}$$

where $Pr_{r_\mathbf{y} \mid r_f}\left(r_i \mid r_j\right)$ is a discrete distribution defined thus with $\alpha \in (\frac{1}{2}, 1)$:

r_j, r_i	-1	0	1
-1	α	$(1-\alpha)/2$	$(1-\alpha)/2$
0	$(1-\alpha)/2$	α	$(1-\alpha)/2$
1	$(1-\alpha)/2$	$(1-\alpha)/2$	α

Note that sampling from $\rho_{\mathbf{r}_{\mathbf{Y}_t} \mid \mathbf{r}_{f_t,\mathbf{X}_t}}$ in (9.31) produces a vector of relationships, not locations. The use of a product in (9.31) implies mutual independence between elements in $\mathbf{r}_{\mathbf{Y}_t}$, so sampling a vector of such relationships is trivially accomplished by taking multiple independent samples from $Pr_{r_\mathbf{y} \mid r_f}$. Details are given in Algorithm 6, a straightforward adaptation of Algorithm 5 for the bowl-shaped class.

Because this class operates on pairs of samples, EVSI is also defined over test pairs: $EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^? \mid \mathbf{X}_{t-1},\mathbf{Y}_{t-1}}$. The resulting graph is therefore three-dimensional and the comparision between EVSI and EU functions is no longer perfectly direct because EU is not concerned with exploratory tests (pairs) but with exploitative decisions



(a) View of the ridge

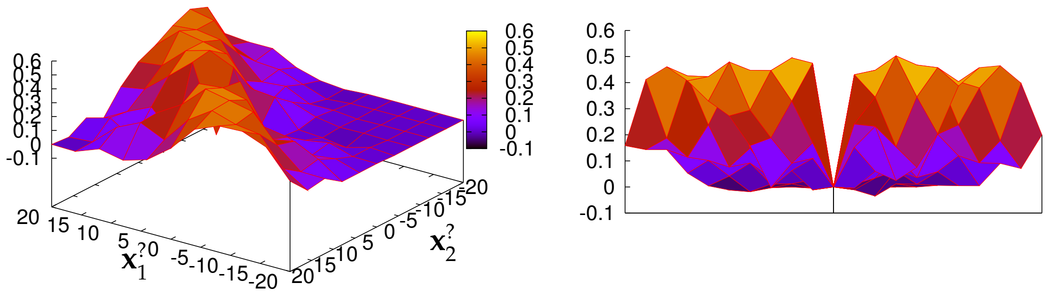
(b) View of the canyon

Figure 9.9: $EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^?}$ on the bowl class at $t = 0$

(singletons); it is clear, however, that MEU will dictate that sampling occur near the assumed location of $\mathbf{x}^\star$. In a pair-wise test scenario such as this, MEU will therefore generally choose to sample the same point twice without the artificial introduction of noise.

The result of running EVSI on this bowl-shaped class at $t = 0$ is shown in Figure 9.9. The true function is again a one-dimensional cone centered at 5. Figure 9.9(a) shows a viewpoint of the $EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^?}$ function that emphasizes its ridge. The location and orientation of the ridge is interesting, indicating that points located on opposite sides of the assumed location of the global maximum (as dictated by the prior) are favored over points found on the same side of it.

The rotated viewpoint in Figure 9.9(b) highlights a canyon that cuts through the middle of the EVSI plot. This valley of low EVSI values occurs when the pair consists of two copies of the same location, a test that will not provide useful information for this class. That the results are so sensible is significant: EVSI discovered this pattern without any more information than a straightforward declaration of the function class and the basic definition of exploitative utility in (9.3).



(a) View of the ridge

(b) View of the canyon

Figure 9.10: $EVSI_{\mathbf{x}_1^?,\mathbf{x}_2^? \mid \mathbf{X}_0,\mathbf{Y}_0}$ on the bowl class at $t = 1$

Again, MEU indicates that the opposite should be done: sample near the maximum and never deviate from it. EVSI not only corrects that problem, but also maximizes the benefit of sample points by ensuring that they occur on opposite sides of the maximum, thus helping to differentiate between functions more effectively; points on the same side of the maximum merely indicate that the true maximum is somewhere to the right or left of them, where points on opposite sides can narrow the search space substantially.

At $t = 1$, EVSI has some evidence available for its calculations, and the result of its application is shown in Figure 9.10. The canyon is again evident: composing a pair of points by copying a single location fails to provide any information in this relationship-oriented class. Predictably, the ridge has shifted from its location at $t = 0$, indicating that the true global maximum is better known with the availability of additional data and that samples should still fall on either side of it. Observe also that the absolute magnitude of EVSI has decreased in the presence of evidence, essentially indicating that less value improvement is expected with every new piece of information: when more knowledge is acquired, additional knowledge is less valuable.

It is important to emphasize that the basic EVSI algorithm is unchanged. The only difference between this and the previous example is the function class. Similar to the previous example, it would be possible to develop a reasonable heuristic for this class that obviates the need for EVSI, but two things are important to mention in this regard: first, without the use of EVSI the nature of an appropriate heuristic would be unclear; and second, EVSI adapts rationally to the specified function class, whatever it may be.

Because the algorithm is essentially statistical inference and expectation calculations, it will perform as well as possible given the supplied information. If any other algorithm consistently finds information about the global optimum in fewer function evaluations, it is operating on assumptions that have not been supplied to the UFO.

The use of EVSI in the UFO is a powerful way of determining how to optimally allocate trial samples for optimization. As described thus far, however, it fails to answer an important question for the empirical optimization problem: when exploration should cease. At some point, the solution provided by any optimization algorithm must be accepted as "good enough", but without a notion of sample cost it is impossible to say when that point is reached with any certainty. Not surprisingly, EVSI provides the tools to answer this question; the oil drilling example makes this clear because the end result was an indication that the test was not cost-effective, knowledge that was obtainable because the cost of the test was supplied. Defining cost in the optimization setting allows for a similarly principled halt to exploration.

9.7 Samples Are Not Free

Rational sample locations are computed by EVSI in the UFO, provided accurate (and intuitive) definitions of the following:

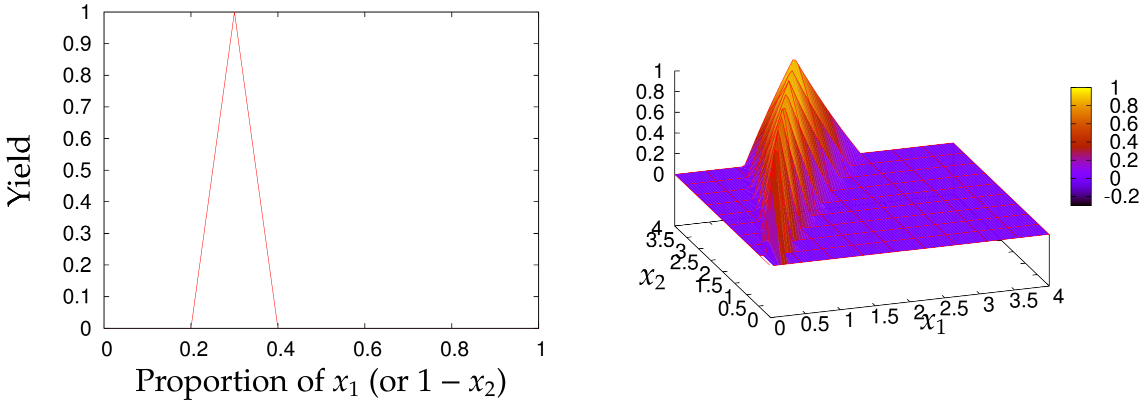
- The function class,
- The utility of a given output, and
- The cost of a particular test.

The first item received detailed attention in previous work [Monson and Seppi 2006], the second has been emphasized in this work, and the third will be developed here as part of the larger context of using UFO as a unified system; all specifications will be outlined for an optimization scenario, showing how a practitioner might approach it using the UFO while introducing the notion of sampling cost.

Consider a laboratory technician tasked with finding an optimal mixture of chemicals, where optimality is achieved by maximizing the percentage yield of a precipitate. For each discovered yield there is a proportional commission, and the technician must purchase the ingredients for the mixture. In this situation, several things are immediately apparent:

- The important data in the experiment is relative; proportions are more important than absolutes.
- Measurement noise is a stark reality in the world of chemical experiments, present at both the inputs and the outputs of the process.
- The output of the experiment has measurable utility.
- Ingredients that make up the starting solution have known cost.
- Mixing a solution is generally a one-way process; it is not possible to remove individual chemicals from a mixed solution.

The UFO provides ways in which all of these concepts can be intuitively described so that experiments are performed rationally. Appropriate application of EVSI, given the function class of interest and the above information, will ideally tell a lab technician how to maximize information while minimizing cost. The



(a) Proportion Coordinates

(b) Absolute Coordinates

Figure 9.11: The truncated cone function

setup required to incorporate the above information into the UFO so that useful results are obtained is outlined here.

9.7.1 Step 1: Define the Function Class

Assume that for this problem the function class is very simple: the mixture consists of exactly two ingredients, and the percentage yield is defined as a truncated cone in the proportion space (Figure 9.11). While more complex classes are possible and even desirable, this class is simple enough to serve the purposes of pedagogy while being difficult enough to illustrate the exploratory behavior of the algorithm. The class contains the true function by leaving some of its parameters undefined, thus:

$$f^{\star}(\mathbf{x}) = \max\left\{0, \left\| 10\left(\frac{\mathbf{x}}{\sum_{i=1}^{D} x_i} - \mathbf{c}\right)\right\|_2\right\} \tag{9.32}$$

where $x_i, c_i > 0$ and $\sum_{i=1}^{D} c_i = 1$. The true function $f^{\star}$ is a member of this class with $\mathbf{c} = (0.3, 0.7)^{\top}$.

The prior is defined similarly to (9.5), using a parameter distribution $\rho_{\mathbf{c}}$, which in this case is uniform in the proportion space. The optimization distribution

$\rho_{\mathbf{x}^\star|f}$ is the same as that in (9.8): a delta function on the parameter vector $\mathbf{c}$. This means that $\mathbf{x}^\star$ is a proportion vector, and the desired result is a set of proportions, not absolute amounts of constituent ingredients.

9.7.2 Step 2: Define the Sampling Distribution

The sampling distribution $\rho_{\mathbf{Y}|f,\mathbf{X}}$ can be used to indicate the presence of sampling noise or subjective uncertainty. In the presence of noise, the distribution indicates that multiple samples at the same location will produce different values over time, and in the case of subjective uncertainty it is an indicactor of a level of confidence in the amount of meaning that can be attached to the samples [DeGroot 1970]. Naturally, both may coexist in the same distribution.

Previous definitions of $\rho_{\mathbf{Y}|f,\mathbf{X}}$ have incorporated noise as a representation of uncertainty, but in this example real measurement noise is present, both in measuring constituents for the solution and in measuring the yield. Noise may also represent environmental factors that are beyond the control of the technician.

Measurement error, which is assumed to dwarf any other noise in this setting, is dependent upon the absolute quanitity of both the individual ingredients and the resulting substance. Because both of these are manifest in output noise, only output noise will be expressed: as the amount of input substance approaches zero, the difficulty of measuring proportions accurately increases rapidly. Measurement noise, therefore, should become greater as quantities become very small.

Equally difficult, however, is accurate measurement of proportions involving very *large* quantities: vats, lakes, oceans, or entire planets of ingredients; as the absolute measurements of consitutent ingredients increases beyond a certain point, the noise will also increase. A function which suitably describes this sort of behavior is the scaled and shifted log-gamma function shown in Figure 9.12.

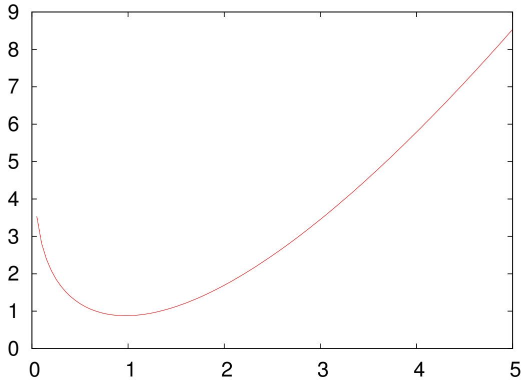


Figure 9.12: $1 + \ln\Gamma(3x/2)$: Suitable for $\sigma_{\mathbf{y}}$ scaling

To be more precise, the sampling distribution is Gaussian with a standard deviation that is scaled by the scaled and shifted log-gamma function:

$$\rho_{\mathbf{y}\,|\,f,\mathbf{x}}(\mathbf{y}\,|\,f,\mathbf{x}) = N\left(f(\mathbf{x}), \sigma_{\mathbf{y}}\left(1 + \Gamma\left(\frac{3}{2}\frac{1}{D}\sum_{i=1}^{D} x_i\right)\right)\right) \quad . \tag{9.33}$$

This definition assumes that the units of the ingredients have been scaled so that 1 unit is most accurately measurable. It is also assumed that accuracy is a function of the average of the constituent quantities.

The details of the Gamma function and the way that noise is added are shown here to highlight the fact that noise can be expressed as exactly what it is; there is no need to develop a heuristic that makes an optimization algorithm tend toward values of high accuracy, since the *reasons* for that heuristic are actually due to noise. The UFO allows for expressions of concepts in their natural form, without an awkward transformation into a heuristic.

9.7.3 Step 3: Define the Utility of Output

Utility, as previously stated, is a simple function of the percentage yield of a given experiment: the technician receives a commission for the yield discovered by each experiment performed, say \$1 × %Yield:

$$u_{f_t,\mathbf{x}_t}(f_t, \mathbf{x}_t) = \$ f_t(\mathbf{x}_t) \ . \tag{9.34}$$

This system of rewards would not necessarily motivate a lazy technician to find good values, since it is assumed that payoff occurs at the end of every experiment. It can easily be extended, however, and will be shown to perform very well despite its simplicity.

Even though the goal of the technician is to *maximize information* about regions of higher payoff, that goal is not directly reflected in the utility function. Instead, utility is defined in terms of exploitation.

9.7.4 Step 4: Define the Sampling Cost

The final required specification is the sampling cost. Whether the true function is queried in simulation or in the laboratory, sampling it has an associated cost. In simulation that cost is often measured in units of time, where in this example it is a dollar amount related to the cost of the constituent ingredients. Whatever the case, cost always exists and is often easily quantified.

In this example, an obvious and trivial way to define sampling cost is as the sum of the prices of the constituent ingredients that go into an experiment:

$$c = \sum_{i=1}^{D} c_i x_i \tag{9.35}$$

where c_i is the cost of a single unit of ingredient i. This is a straightforward definition of cost from the technician's perspective, who is required to buy his own ingredients.

When mixing a solution in order to achieve a precipitate, however, it is often possible to add ingredients *incrementally* to adjust the percentage yield, even after some amount of precipitate has been previously removed and measured. In other words, it is possible to perform a new experiment by *continuing an old one*, giving rise to a more interesting cost function:

$$c_t = \begin{cases} \sum_{i=1}^{D} c_i x_{t,i} & \text{if } \exists i.\ x_{t,i} < x_{t-1,i} \text{ or } \mathbf{x}_t = \mathbf{x}_{t-1} \\ \sum_{i=1}^{D} c_i \left(x_{t,i} - x_{t-1,i}\right) & \text{otherwise} \end{cases} . \tag{9.36}$$

If the technician desires to reduce an ingredient or to duplicate an experiment, the cost is calculated by totalling the cost of the constituents; the only way to repeat or reduce constituent quantities is to begin again. If, on the other hand, he wishes to adjust the balance by adding a small amount of something, the cost is measured as the price of the increased ingredients, not the price of the full solution.

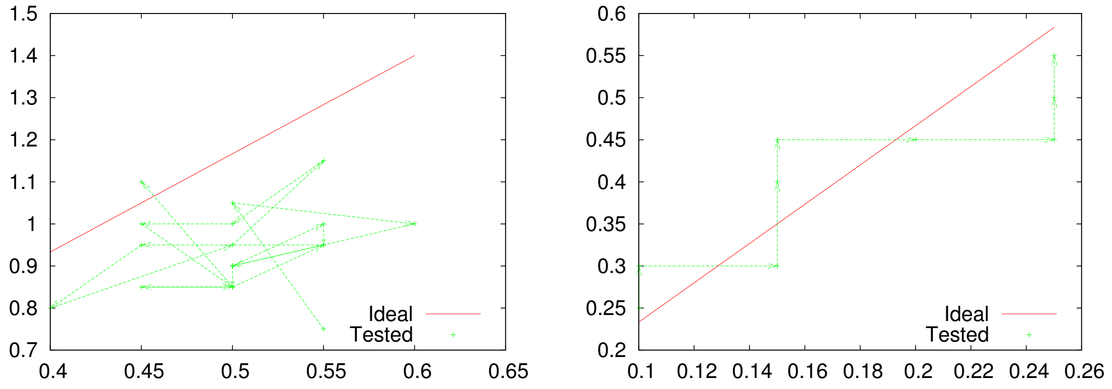
This definition of cost describes the nature of the world in a straightforward way. Again, a heuristic could be developed that encourages an optimization algorithm to move forward in the space of quantities, but the UFO is purely declarative and does not require such heuristics; it merely wants to be told the truth about the way that the world behaves, and this cost function is an example of such a straightforward specification.

9.7.5 Step 5: Profit! (Behavior and Results)

The results of using EVSI for experiment selection in this section assume the following:

- Sampling noise has a base standard deviation of $\sigma_{\mathrm{y}} = 0.1$,
- All ingredients cost \$0.10 per unit, and
- The best proportion is $\mathbf{c} = (0.3, 0.7)^{\top}$.

Figure 9.13 illustrates the path taken by EVSI through the space of absolute ingredient measurements both with and without consideration of cost: Figure 9.13(a)



(a) No cost

(b) Mixture cost

Figure 9.13: Recipe experiments, with and without cost included

shows the path of experiments when EVSI is not aware of the sampling cost, and Figure 9.13(b) illustrates the path when costs are supplied. In the first case, it continues experimenting until the maximum number of iterations has been reached (20 for this example) and does not appear to be following any particular pattern. In the second, it not only always adds ingredients incrementally to the mixture, it adds them *one at a time* and *stops after* 9 *iterations*.

This demonstrates that the inclusion of cost not only admits a sane sampling policy, it also creates a natural stopping criterion: when sampling cost exceeds EVSI, then sampling ceases. Not only did the UFO dictate when to stop, in this example it also selected experiments that tended to minimize cost: the technician that did not provide a cost function to EVSI spent $2.22, while the technician that wisely included a straightforward cost definition spent only $0.08 on ingredients. Even if they had both stopped after 9 iterations, the first technician would have spent $1.17 with negligible difference in estimated proportion quality; after 9 iterations, both would receive similar commissions by exploiting what they know.

Making EVSI aware of the cost of ingredients causes the UFO to behave rationally in sophisticated ways. It selects experiments based on a seemingly complex mixture of potential utility and sampling cost, and it does so in such a way that the result is what a human would call rational: it favors incremental addition of ingredients while avoiding measurements that are too small or too large. It also defines when exploration should stop and exploitation begin. That it does so with intuitive declarations about the nature of the environment is compelling.

9.8 When Exploration Precludes Exploitation

Once the chemistry experiments have been completed and a distribution over good proportions discovered in a laboratory setting, the process is taken to production level volumes, where higher volumes of ingredients are mixed together to obtain larger sellable output. In this setting, if changes are made to the production mixer, they must be made directly to the proportions, assuming that the plant is already operating at capacity.

Because transfer from the lab to production is never perfect, it is desirable to continue learning about the optimal point in this higher-volume setting. Changes in this setting, however, cannot be made with impunity; once the system goes into production, it starts to produce a reliable, if not optimal, revenue stream. The risks of causing a drastic short-term drop in production in the quest for slightly better long-term output are significant.

EVSI can be used to continue the learning process in production while mitigating this risk. What is needed is a definition of cost that takes into account lost revenue due to experimentation on the production equipment: *opportunity cost*. Intuitively, opportunity cost is simply a measure of the difference between what revenue might have been achieved through exploitation (MEU) and the revenue

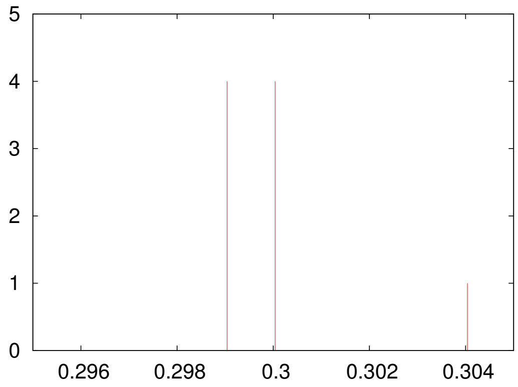


Figure 9.14: Distribution of points selected when employing opportunity cost

actually achieved during exploration of the possibly suboptimal parameters $\mathbf{x}_t^?$:

$$c(\mathbf{x}_t^?) = \max_{\mathbf{x}_t} \mathrm{E}_{\mathbf{x}_t \mid \mathbf{e}_t} \left[u_{\mathbf{x}_t, f_t} (\mathbf{x}_t, f_t) \right] - \mathrm{E}_{\mathbf{x}_t^? \mid \mathbf{e}_t} \left[u_{\mathbf{x}_t, f_t} \left(\mathbf{x}_t^?, f_t \right) \right] \quad . \tag{9.37}$$

This formulation of cost is convenient in many ways, including the fact that its units are the same as those of EVSI: the difference of two utilities. EVSI measures the expected *improvement* in utility given extra information obtained from a test sample $\mathbf{x}_t^?$, and opportunity cost measures the expected *loss* given that the plant is not maximizing expected utility for the duration of the experiment.

When the expected gains of exploration fail to outweigh the expected loss associated with suboptimal operation, the algorithm stops producing new values for search, indicating that no amount of exploration is appropriate. Application of opportunity cost to this example results in the selection of a different distribution of samples than that previously seen, shown in Figure 9.14. The samples were obtained after running the opportunity cost algorithm 200 times; only 9 of the 200 trials chose to do any sampling at all, and those that did explore did so only once near the known maximum, making it behave more like MEU while still doing some exploration.

Once it is clear that exploration should cease, MEU is an appropriate way to exploit the information gained earlier. If at any time the cost or utility functions change, then opportunity cost may be used in the UFO to perform judicious exploration, and the operator is ensured such exploration will be conservative, stopping when all possible benefits have been exhausted. That this can be done at all is a testament to the UFO model; without its explicit information relationships, the application of EVSI and straightforward use of cost would be impossible.

9.9 Conclusions and Future Work

The specifications that the UFO requires, i.e.,

- The class of functions of interest (ρ_f and $\rho_{f_t \mid f_{t-1}}$),
- The nature of sampling noise ($\rho_{\mathbf{Y} \mid f, \mathbf{X}}$),
- The intent of optimization ($\rho_{\mathbf{x}^\star \mid f}$),
- The value of results ($u_{f_t, \mathbf{x}_t}$), and
- The cost of samples ($c(\mathbf{x}_t^?)$)

are reasonable requirements. They are all straightforwardly declared and readily available to an optimization practitioner, with the possible exception of the function class (addressed elsewhere [Monson and Seppi 2006]). That the specifications are simply and intuitively specified does not limit the sophistication of the resulting optimization algorithm. In fact, as illustrated in preceding examples, the behavior can be very sophisticated indeed, all while the underlying optimization algorithm remains fixed.

Traditionally such examples would be used to overwhelm the reader with data, perhaps providing strong empirical evidence that the presented algorithm is better than another. In this case, however, the examples have been used to show that *predicted behavior* aligns compellingly with *observed behavior*. The UFO is significant precisely because it allows accurate predictions to be made (e.g., that maximizing EVSI tends to produce a more confident and accurate posterior than MEU); no other method provides a principled way of doing so. Indeed, it can be mathematically argued that the UFO is an optimal optimizer with respect to the information it has, a proof that it inherits from its use of Bayesian inference and information value theory [DeGroot 1970]. If another approach consistently outperforms the methods outlined here, then it must have more information at its disposal.

The use of a statistical model as the basis for optimization allows all available information to be used during the search for a global optimum. This is in contrast with traditional evolutionary optimization algorithms, where they can either be applied to a particular problem or not, and tuning is generally not easy to do in a principled way. The use of such algorithms is often an exercise in costly guesswork, either in finding the right algorithm for a particular problem or in discovery of appropriate algorithm parameters. Increasing the sophistication of such algorithms, e.g., to make them behave rationally in the presence of known costs, is never a simple exercise and results in human-developed heuristics that must then be tested to determine whether they exhibit subtle and unexpected behavior. When additional information is made available, this exercise must be repeated, often with mixed results.

Additionally, when an existing algorithm is discovered that appears to be a good match to the problem at hand, it does not yield much new information about the true nature of the function; all that is known is that it is a better match than any of the other algorithms tried. In contrast, the use of the UFO enables true information to be gained about an unknown function. Instead of selecting from a toolbox of opaque algorithms, it enables selection of transparent function class definitions; success using one of these definitions indicates that the function is likely to be a member of the corresponding class, potentially providing a wealth of information about its true nature. That information can also be used to compare function classes in a principled way because of the Bayesian framework in which optimization is performed [MacKay 1992a,b], an idea that should be pursued in future work.

The UFO also admits the natural expression of the various stages of real-world optimization: laboratory experimentation, careful production exploration, and simple exploitation. The direct determination of whether to explore or exploit is no longer the question that practitioners face; instead they must simply define what it costs to sample and what the obtained values are worth. The UFO then determines not only how to explore, but when to stop exploration and switch to exploitation.

Several other concepts are expressable in this model that have not been covered in detail in this work. One may, for example, incorporate a notion of "risk-seeking" or "risk-averse" behavior into the algorithm by simply employing standard utility-altering tricks (e.g., squaring or taking the square root of the utility function). One may also incorporate such notions as the time-value of money, the fact that an experiment can lose money over time if the difference is made up over a different period of time, etc. All of these are specifications that find clear expression within the utility and cost functions, and make interesting avenues for future study.

The determination of the function class remains the largest hurdle in the use of the UFO, but many of those issues have been discussed and addressed in previous work [Monson and Seppi 2006]: determination of an appropriate function class is in many ways no more taxing than trying to select the right algorithm for the job. There is certainly room for more standard function class definitions, a potentially fruitful area for future research.

The UFO is not, of course, a panacea. There are real computational costs associated, for example, with EVSI computation. The complexity of EVSI, for example, is usually substantially higher in terms of raw CPU cycles than that of many popular evolutionary algorithms; it is cubic in the number of functions sampled from ρ_f, and that number should generally be fairly high to ensure good coverage of the distribution's support space. As the number of dimensions increases, that number should increase further, making scalability an issue. It is likely, however, that mathematical approximations may be applied to alleviate these problems [Brennan and Kharroubi 2005]. Additionally, only particle filters have been applied to this model so far, but other empirical inference methodologies exist that may be more efficient in this setting.

The EVSI calculation returns an answer for a particular test $\mathbf{x}_t^?$, which must be chosen outside of the algorithm. The test that yields the greatest positive EVSI value is generally the one that is used, suggesting that the algorithm for choosing a sample point involves choosing a test which maximizes EVSI: an embedded optimization problem. It is possible that bootstrapping might be applied to make the choice of candidate tests more principled. This interesting idea is in keeping with the tradition of Bayesian statistics in general; frequently a good prior is not available and one must be generated from existing data and a higher-level set of assumptions. While this can become absurdly recursive, the idea is generally applied in moderation.

The UFO provides a way of thinking about the problem of optimization that is both novel and powerful, as well as suggesting an existing and well-studied set of solution methodologies. The calculations of EVSI and expected utility are well understood and often applied in other settings. Their use in the context of optimization is both natural and significant, providing a set of tools to the practitioner that not help to achieve the goal of optimization while supplying information about the nature of the problem.

Chapter 10

Conclusion

This work has been primarily concerned with the problem of continuous, unconstrained, single-objective optimization where sparse information about the target function is obtainable but important information remains unknown. An understanding of the practical issues presented by No Free Lunch and a statistical perspective on the mechanics of PSO laid the groundwork for a more general and principled approach to optimization through the use of Bayesian reasoning. This approach achieves the goal defined in the thesis statement of this work: to create a principled decision-theoretic model of optimization that addresses many of the issues posed by No Free Lunch.

Arriving to the core contribution of this work was a process of learning and refinement that is evident in the progression of individual papers that comprise the whole. Each of these constituent papers has drawn adequate conclusions of its own; those conclusions that are most relevant are collected and summarized here.

10.1 No Free Lunch

All of the papers in this work present concepts that are at least influenced by No Free Lunch even when not addressing it directly: Part I develops various analytical or algorithmic domain-specific improvements to PSO, each paper serving as an example of the need for an explicit definition of the limited function class that

NFL dictates must be present; Part II applies Bayesian reasoning to PSO motion then shows that this same reasoning can explain existing behavior, simultaneously improving understanding of why PSO works and effectively demonstrating how little is known about the function classes corresponding to the several PSO variants; and finally, Part III addresses NFL directly, bridging the traditional gap between design and performance desiderata by making the function class an explicit part of an algorithm-generating model of optimization. The impact and importance of No Free Lunch is evident throughout.

Chapter 8 makes it particularly clear that thinking about optimization in terms of an explicit statistical model not only grants NFL more visibility in the problem of optimization, but also allows it to be examined more directly in the continuous context. The proposed Bayesian optimization network establishes and clarifies the following:

NFL exists: It has been known for some time that all optimization algorithms are equal when considering all possible functions, but application to the continuous domain has been awkward. This work, while not supplying rigorous proofs, provides the tools for such proofs by presenting a model that makes application of NFL to continuous settings obvious.

Optimization problems have distinct complexities: That optimization problems can be ranked according to some complexity measure is not new information, but the model presented here makes it clear precisely what that measure should be in the continuous setting. This work suggests that both *computational complexity* and *sample complexity* play a role in the difficulty of a given optimization problem, and it provides tools for their explicit calculation.

Representation is a key to problem complexity: This concept has been known in the machine learning community for some time: no learning algorithm's performance can be considered without due attention to the features it is given and the representation it uses to learn them; some representations are better than others for the same problem. Current proofs of the existence of optimization complexity classes take an algorithm as an axiom and then show that not all problems are equally difficult for it [Macready and Wolpert 1996]. In contrast, this work establishes that it is not the *algorithm* that allows problems to be ranked according to complexity, but the problem *representation*; that the traditional proof works at all is due entirely to the fact that every algorithm at least embodies an implicit representation of the problem.

10.2 Bayesian Inference

The Bayesian models of optimization allow for intuitive expression of function characteristics that have traditionally been difficult to incorporate into existing algorithms. In particular, they allow for the representation and optimization of functions that change over time, functions whose outputs are truly nondeterministic, and potentially functions with multiple simultaneous optimization goals.

Functions whose outputs are noisy (i.e., each sample at a single location yields different results over time, according to some distribution) tend to be difficult for many optimizers, and are often simply excluded from consideration in the evolutionary computation community. PSO, for example, has a greedy policy that creates and maintains particle attractors based on the quality of their corresponding values in reference to what has been seen in the past; overly optimistic samples will cause the swarm to be attracted to unproductive locations of the search space. Most existing evolutionary algorithms have the same problem in various forms, and noise is typically addressed as an afterthought.

A similar situation exists with dynamic functions, though these have received a great deal more attention in the evolutionary computation literature; they are, after all, easier to attack using heuristic approaches like periodic rerandomization. Such strategies can be effective, but display the same weakness as all evolutionary algorithms: their design is accomplished using heuristics that do not directly describe the dynamic functions on which they are intended to work.

The Bayesian approaches outlined here, however, take noisy and dynamic function into account directly and naturally. Because they represent a statistical view of the optimization problem with explicit inclusion of the function class, noisy outputs and time-dependent parameters are already part of the model and can be called upon whenever needed.

Finally, the application of Bayesian reasoning to optimization problems admits a declaration of subjective human confidence. In return, inference in the network produces a distribution over possible optima, and the shape of that distribution may be viewed as an expression of confidence that adapts to acquired information. While this idea is not new to those conversant with Bayesian reasoning, its application to optimization sheds new light on how confident a practitioner may be that the results obtained from a few samples actually point to the location of the global optimum.

10.3 Utility

That optimization results can have associated utility is perhaps the most significant contribution of this work. Not only is NFL clarified and made quantifiable while allowing all kinds of functions and search goals to be expressed, but all of the practical information at the disposal of an optimization practitioner can be supplied in its natural form to a statistical, utility-based algorithm.

People performing real-world optimization generally do so because optimality has a concrete associated benefit, e.g., they are trying to maximize dollars or minimize accidents. The very goal of optimization, whatever the function, is to learn how to increase benefits without incurring unreasonable costs, and the Bayesian model presented here acts rationally in the presence of that information.

No other optimization approach incorporates so much information in such a natural way. It is possible to make use of utilities or costs with existing algorithms, but again this must typically be done by creating and employing heuristics that directly affect algorithm behavior, something that cannot be done lightly because existing algorithms generally confound all aspects of their behavior with their implicit function class definition. This model not only separates declaration of the function class from definitions of cost and utility, it allows them to be specified in their naturally occurring form.

These declarations also provide the means of answering a question that plagues every optimization researcher: when should sampling stop? The Bayesian utility model not only acts rationally while sampling the function, it also knows when it is rational to abandon sampling altogether, providing a much-needed stopping criterion to empirical optimization while transforming traditional explore/exploit tradeoff specifications into a simple declaration of utility and cost.

10.4 Directions for Future Research

While the contributions of this work are exciting in their own right, perhaps more exciting is the fact that it has generated more ideas than can be immediately pursued. Some of these follow.

Connection to existing approaches: The algorithm in Chapter 8 can be connected to PSO with an appropriate choice of function class and some careful approximations, perhaps indicating that PSO can be viewed as an approximate but motivated statistical approach. This may apply to other existing algorithms, providing insights into their behavior and function classes.

Creation of function class definitions: That the function class must be defined appears to be the weakest characteristic of the Bayesian approach, requiring more of practitioners than has apparently been necessary in the past. The requirements are not, in fact, more difficult, but it is true that they have changed: instead of creating an algorithm to attack a particular problem, one must accurately define the problem's characteristics. At least one interesting function class has been introduced, showing that it may not be as difficult as initially thought, but a larger toolbox of such classes is needed.

Choice of approximate inference algorithm: The use of a particle filter as the inference method in the network is not arbitrary, but is not likely to be the most efficient choice, either. Many such methods exist and should be explored.

EVSI approximations: EVSI calculations are fairly time-consuming when compared to the rest of the methods employed, and research has been done to attempt to speed them up through approximation [Seppi 1990; Brennan and Kharroubi 2005]. Such approaches should be tested in the optimization context to assess their behavior and utility.

The model of optimization presented here provides interesting and exciting new ideas that not only answer some difficult questions, but point the way toward other questions that may be even more interesting. The questions that the utile optimization model does address are important and long-standing, e.g., where best to sample, when to stop, and what to do with the obtained information; it does this with an explicit, clear, and often intuitive declaration of the nature of the environment in which optimization occurs, allowing practitioners to obtain a working, rational algorithm in return for simply specifying what they know. Much remains to be studied in this area, and there is great potential for increased understanding and more principled approaches to the pervasive problem of optimization.

Bibliography

C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973:420–434, 2001.

P. J. Angeline. Using selection to improve particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, pages 84–89, 1998.

K. Arrow. A difficulty in the concept of social welfare. *The Journal of Political Economy*, 58(4):328–346, 1950.

S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University, 1994.

S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of the International Conference on Machine Learning (ICML 1995)*, pages 38–46, 1995.

P. L. Bartlett, P. M. Long, and R. C. Williamson. Fat-shattering and the learnability of real-valued functions. *Journal of Computer and System Sciences*, 52(3):434–452, 1996.

T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 19–26, 2002.

A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

A. Brennan and S. A. Kharroubi. Efficient computation of partial expected value of sample information using bayesian approximation. Technical Report 560/05, Department of Probability and Statistics, University of Sheffield, 2005. Submitted to Journal of Health Economics.

S. Christensen and F. Oppacher. What can we learn from no free lunch? a first attempt to characterize the concept of a searchable function. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 1219–1226, 2001.

N. Christianini and J. Shawe-Taylor. *An Introduuction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

M. Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1951–1957, 1999.

M. Clerc. TRIBES - un exemple d'optimisation par essaim particulaire sans paramètres de contrôle. In *Optimisation par Essaim Particulaire (OEP 2003)*, 2003.

M. Clerc. Math stuff about PSO. Online at `http://clerc.maurice.free.fr/pso/`, 2004.

M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

G. Coath and S. K. Halgamuge. A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 2419–2425, 2003.

J. S. de Bonet, J. Charles L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9:424–431, 1997.

J. F. G. De Freitas, M. A. Niranjan, A. H. Gee, and A. Doucet. Sequential monte carlo methods to train neural network models. *Neural Computation*, 12(4):955–993, 2000.

M. H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.

R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2000)*, pages 84–88, 2000.

R. C. Eberhart and Y. Shi. Particle swarm optimization: Developments, applictions, and resources. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001)*, pages 81–86, 2001.

V. V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

D. K. Gehlhaar and D. B. Fogel. Tuning evolutionary programming for conformationally flexible molecular docking. In *Evolutionary Programming*, pages 419–429, 1996.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

J. J. Grefenstette. Deception considered harmful. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, 1993.

J. Holland. Genetic algorithms and the optimal allocation of trials. *SICOMP*, 2(2): 88–105, 1973.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

J. Horn. Genetic algorithms, problem difficulty, and the modality of fitness landscapes. Master's thesis, University of Illinois at Urbana-Champagne, 1985.

X. Hu and R. C. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics 2002 (SCI 2002)*, pages 203–206, 2002.

C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3:313–322, 2004.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497, 1994.

J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and Z. Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, pages 1931–1938, 1999.

J. Kennedy. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2000)*, pages 1507–1512, 2000.

J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 80–87, 2003.

J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *International Conference on Neural Networks IV*, pages 1942–1948, 1995.

J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics, and Informatics*, pages 4104–4109, 1997.

J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, pages 1671–1676, 2002.

J. Kennedy and R. Mendes. Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. In *Proceedings of the 2003 IEEE SMC Workshop on Soft Computing in Industrial Applications (SMCia03)*, pages 45–50, 2003.

J. Kennedy and W. Spears. Matching algorithms to problems: An experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, pages 78–83, 1998.

S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

T. Krink, J. S. Vestertroem, and J. Riget. Particle swarm optimisation with spatial particle extension. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1474–1479, 2002.

P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization by learning and simulation of bayesian and Gaussian networks. Technical Report EHU-KZAA-IK-4/99, Department of Computer Science and Articial Intelligence, University of the Basque Country, 1999.

P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms : A New Tool for Evolutionary Computation*, volume 2. Kluwer Academic Publishers, 2001.

C. T. Leondes, editor. *Theory and Applications of Kalman Filtering*. Number 139 in AGARDograph. North Atlantic Treaty Organization, Advisory Group for Aerospace Research and Development, 1970.

D. V. Lindley. *Making Decisions*. John Wiley and Sons, 2nd edition, 1985.

J. Liu and M. West. Combined parameter and state estimation in simulation-based filtering. In *Sequential Monte Carlo Methods in Practice. New York*, pages 197–223. Springer-Verlag, New York, 2001.

M. Løvbjerg. Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. Master's thesis, Department of Computer Science, University of Aarhus, 2002.

D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992a.

D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992b.

W. G. Macready and D. H. Wolpert. What makes an optimization problem hard? *Complexity*, 5, 1996.

R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural network training. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002)*, pages 1895–1899, 2002.

R. Mendes, J. Kennedy, and J. Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 88–94, 2003.

R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd edition, 1996.

T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

C. K. Monson and K. D. Seppi. The Kalman swarm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 140–150, 2004.

C. K. Monson and K. D. Seppi. Bayesian optimization models for particle swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 193–200, 2005.

C. K. Monson and K. D. Seppi. The evolutionary optimization DBN. *Evolutionary Computation*, In Review, 2006.

E. Ozcan and C. K. Mohan. Particle swarm optimizaton: Surfing the waves. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1939–1944, 1999.

U. Paquet and A. P. Engelbrecht. A new particle swarm optimizer for linearly constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 227–233, 2003a.

U. Paquet and A. P. Engelbrecht. Training support vector machines with particle swarms. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2003)*, pages 1598–1603, 2003b.

K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In *Proceedings of the Euro-International Symposium on Computational Intelligence 2002*, pages 214–220, 2002.

M. Pelikan and D. E. Goldberg. Hierarchical problem solving by the bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 267–274, 2000a.

M. Pelikan and D. E. Goldberg. Research on the bayesian optimization algorithm. In *Optimization By Building and Using Probabilistic Models*, pages 216–219, 2000b.

M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 525–532, 1999.

M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21 (1):5–20, 2002.

G. T. Pulido and C. A. C. Coello. A constraint-handling mechanism for particle swarm optimization. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC'2004)*, pages 1396–1403, 2004.

M. Richards and D. Ventura. Dynamic sociometry in particle swarm optimization. In *International Conference on Computational Intelligence and Natural Computing*, pages 1577–1580, 2003.

J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer — the ARPSO. Technical Report 2002-02, Department of Computer Science, University of Aarhus, 2002.

S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

K. D. Seppi. *A Bayesian Approach to Selected Database Issues*. PhD thesis, University of Texas, 1990.

Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, pages 69–73, 1998a.

Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 591–600, 1998b.

Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1945–1950, 1999.

G. Syswerda. Simulated crossover in genetic algorithms. In *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 239–255, 1993.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

J. S. Vesterstrøm, J. Riget, and T. Krink. Division of labor in particle swarm optimisation. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1570–1575, 2002.

M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, 1999.

D. Whitley and J. P. Watson. Complexity theory and the no free lunch theorem. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2006.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Y. Zheng, L. Ma, L. Zhang, and J. Qian. Empirical study of particle swarm optimizer with an increasing inertia weight. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 221–226, 2003.