A Meta-Parallel Evolutionary System for Solving Optimization Problems

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Winard Britt

Certificate of Approval:

A Meta-Parallel Evolutionary System for Solving Optimization Problems

Winard Britt

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 10, 2007

A Meta-Parallel Evolutionary System for Solving Optimization Problems

Winard Britt

Signature of Author

Date of Graduation

Vita

Winard Ronald Britt, son of Ronald H. Britt and Catherine S. Britt, was born September 15, 1982 in Montgomery, Alabama. In 2000, he graduated with Highest Honors from Booker T. Washington Magnet High School in Montgomery. He attended Auburn University in Auburn, graduating Summa Cum Laude and University Honors Scholar with a Bachelor's of Software Engineering in 2004. In that same year, he entered Graduate School at Auburn University.

Thesis Abstract

A Meta-Parallel Evolutionary System for Solving Optimization Problems

Winard Britt

Master of Science, May 10, 2007
(B. SWEN, Auburn University – Auburn, 2004)

69 Typed Pages

Directed by Gerry Dozier

The purpose of the Meta-Parallel Evolutionary System (MPES) is to develop fast, efficient parallel evolutionary systems for function optimization. Given an optimization problem and a set number of nodes available for the computation, the MPES searches for a strong, potentially heterogeneous combination of evolutionary algorithms to coordinate in order to effectively solve a problem. The Evolutionary Algorithms that are utilized in the parallel system are a Particle Swarm Optimizer (PSO), a variety of Genetic Algorithms (GAs), and an Evolutionary Hill-Climber Algorithm (EHC). The subpopulations communicate with each other via a centralized buffer. At a higher level exists the MPES, which uses evolutionary methods in order to discover parameters for effective parallel systems. This methodology provides an immediate benefit in the form of a strong tool to solve the optimization problem. Further, it provides a long-term benefit by identifying a system that has the potential to effectively solve other difficult optimization problems with a similar search space.

Acknowledgments

The author thanks the following:

NASA Marshall Space Flight Center and Michael Tinker for their support of this research.

Dr. Gerry Dozier for patience.

Kelly Price for going to extraordinary lengths to provide sufficient computational resources for simulations.

Marcus Jackson, Brett Remkus, and Tamay Shannon for the extended use of their computers for simulation runs.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TeX (specifically LaTeX) together with the departmental style-file `aums.sty`.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

Nature provides a vast array of processes from which computer scientists can be inspired to create effective problem-solving methodologies. In nature, creatures adapt and change over time through breeding and genetic mutation to become better fit to survive in their environments. The principles of natural selection have guided the development of prolific research in Evolutionary Computation [Spears et al. 1993][DeJong and Spears 1993][Bäck et al. 1997], which is a class of stochastic methods of searching a potentially complex search space by evolving a population of candidate solutions to that search problem. Evolutionary Computation (EC) comprises a wide array of algorithms including (but not limited to) Genetic Algorithms [Golberg 1989][Holland 1975], Evolution Strategies [Schwefel and Rudolph 1995], and Particle Swarm Optimizers [Kennedy and Eberhart 1995]. Evolutionary Computation has been applied to solve a vast array of problems including function optimization [Golberg 1989], space vehicle optimization [Dozier et al. 2006], game-playing [Fogel 1995], and many others [Spears et al. 1993].

However, natural evolutionary processes do not occur in a vacuum. Subpopulations of creatures interact and alter the available pool of genetic information. Cross-breeding between populations may result in new species with greater capability to adapt and survive. Similarly, evolutionary algorithms can be made to run in parallel for a variety of potential benefits, including the possibility of alternative solutions to the problem, better search, higher efficiency (than traditional evolutionary algorithms), and improved robustness [Alba and Troya 1999]. Many different strategies exist for parallelizing evolutionary systems

[Cantú-Paz 1998] ranging from the very straightforward (running multiple evolutionary algorithms at the same time, as in [Shonkwiler 1993]) to complex hierarchies of evolutionary algorithms with imposed topologies and sophisticated rules for interaction. The goal of this work is to develop high-performance systems using multiple subpopulation of potentially varying algorithms cooperating through intermittent, indirect communication with each other.

Developing a Parallel Evolutionary System to effectively solve a problem depends on a number of factors, including what algorithms comprise the system, how those algorithms communicate, and how often they communicate. To this end, a Meta-Parallel Evolutionary System was developed in order to evolve good parameters for the PES similar to the concepts of evolving parameters for non-parallel evolutionary algorithms presented in [Dozier et al. 2006], [Grefenstette 1986], and [Schaffer et al. 1989]. This MPES should be able to evolve strong Parallel Evolutionary System parameters to solve a wide range of difficult problems, even in the cases where there is relatively little *a priori* knowledge. Ulimately, this could be used as a tool to generate parallel systems with substantially less trial and error optimization than alternate methods.

1.1 Goals

The following list enumerates the primary goals of this research:

1. To explore the potential benefits of heterogeneous parallel systems which achieve their heterogeneity not merely through the use of varying parameter settings, but also through the interaction of different types of algorithms.

2. To explore the effectiveness of using a centralized buffer space to perform indirect communication between evolutionary algorithms running in parallel.
3. To create a means for algorithms with differing representation types to coordinate effectively with each other while utilizing a common representation in the shared buffer space.
4. To develop a Meta-Parallel Evolutionary System to discover high-performance settings for Parallel Evolutionary Systems.
5. To empirically test the effectiveness of the aforementioned systems on a set of difficult optimization problems.

1.2 Thesis Outline

This introduction has provided a broad introduction to the overarching goals of the MPES system. The remainder of this thesis is structured as described below.

In Chapter 2, Evolutionary Algorithms are described to the general algorithmic level of detail. Specific categories of algorithms used in this work are given primary focus. Background for Parallel Evolutionary Systems is given, with specific focus on Coarse-Grained Evolutionary Systems, which are the primary thrust of this work.

In Chapter 3, the Parallel Evolutionary System is introduced along with the significant parameters for its operation. The specific implementations of the algorithms developed for the subpopulations are described.

In Chapter 4, the Meta-Parallel Evolutionary System and its variants are introduced and described. Some example candidate solutions for the system are provided and described in detail.

In Chapter 5, the collection of minimization problems chosen to test the MPES system is shown along with the constraints associated with their inputs for the system.

In Chapter 6, experiments on a number of PES and MPES systems are given, along with their associated results and analysis.

In Chapter 7, some concluding remarks are made.

Chapter 2

Background

2.1 Evolutionary Algorithms

The purpose of this section is to provide a basic concept of the workings of each of the evolutionary algorithms utilized in this research.

2.1.1 Genetic Algorithms

A Genetic Algorithm (GA) operates on a population of encoded candidate solutions (called "individuals") to a problem. The specific values (which may be real-coded values or some binary representation) of the individual corresponding to problem are called the "chromosome." Each individual has a corresponding "fitness" value which represents the desirability of the solution. This fitness value is obtained using a fitness function, which maps the chromosome to a numerical value. For example, a GA trying to find the minimum value for the function $f(x, y) = x^2 + y^2$ might have a chromosome of $< 2, 0 >$, where 2 corresponds to x and the 0 corresponds to y. The fitness function would simply evaluate the function itself to yield a fitness value of $2^2 + 0^2 = 4$. In the case of a minimization problem, an individual is said to be more fit if it has a lower fitness value. Fitness functions are problem specific.

In general, the GA operates by creating a population of individuals, evaluating them, choosing some of them to be parents (individuals which will be manipulated in some way to create new individuals), creating some children, evaluating the children, and then finally choosing some survivors from the parents and/or children based on their fitness values.

Figure 2.1: Pseudocode for a Generic Genetic Algorithm

```
T = 0
Initialize population P(t)
Evaluate P(t)
while (best member of P's fitness < desired fitness AND t < some number of iterations)
    select parents
    create offspring from chosen parents
    evaluate offspring
    P(t+1) = survivors of parents and children
```

This process is typically repeated many times until a certain fitness value is achieved or some number of cycles (also called “generations” in the literature) passes. In Figure 2.1, pseudocode for a typical GA is shown. Children are typically created through variation operators, such as crossover (combining attributes from parents) and mutation (making tiny changes to a given value in the chromosome). The central idea of GAs is that highly fit individuals have traits that contribute to their high fitness and will pass on those traits to children. Hopefully, through recombination and mutation, the GA will produce an individual with all the traits of the perfect (optimum) solution. In a sense, the operation of a GA is a careful balance between creating diversity in the solutions (through variation operators) and selection pressure (choosing highly fit individuals to survive). Excessive diversity in a population will tend to cause the GA to have difficulty converging to highly fit solutions. Excessive selection pressure will cause the population to converge rapidly, but often this convergence is to suboptimal solutions.

A number of parameters affect the performance of GAs, such as representation of the chromosome, choice of parents, choice of variation operators, and population size. Different parameters are more successful for certain problems and types of problems than others and

Figure 2.2: Pseudocode for a Generic Hill-Climber Algorithm

```
t = 0
Initialize individual i
Evaluate i
while (i's fitness < desired fitness AND t < some number of iterations)
    choose a step location
    evaluate step location
    if step yields better fitness
        replace i with new individual corresponding to the step
```

it is generally held that no single set of parameters works best for all problems [Wolpert and Macready 1990].

For a more detailed discussion of Genetic Algorithms and issues associated with them, consult [Golberg 1989] or [Davis 1991].

2.1.2 Hill-Climbers

As shown by the pseudocode presented in Figure 2.2, the notion of hill-climbing is fairly straightforward. Start with a candidate solution in the search space and evaluate your fitness (which corresponds to height). Then move in a direction of the steepest ascent (or descent, in the case of a mimization problem). A move or step can be determined by either using a calculus-based method (gradients) or randomly (a variation operator). Hill-Climbers which use randomization are typically referred to as "Evolutionary Hill-Climbers." A potential step is tested and if it yields a better fitness value, the hill-climber moves to that location (this is very similar to a genetic algorithm with a population of one). Steps may be a fixed, variable, or random distance. This process can be repeated until a solution is found or a set number of iterations has passed. Hill-Climbers typically perform well at finding a local best in an area of the search space, but typically perform poorly at finding a global best unless special measures are taken, such as random restarts or randomization [Golberg 1989].

2.1.3 Particle Swarm Optimizers

PSOs are based on the movements of groups of organisms, such as fish or birds [Kennedy and Eberhart 1995]. The “swarm” consists of a group of “particles” (analagous to individuals in the previously mentioned EAs) traveling through the search space. Particles have three vectors (similar to the chromosome in a GA): their current location, the best location that they have discovered so far, and a velocity vector that governs their movement. They also store the fitness of their current location and the fitness of the best location they have seen so far. On each cycle, the particles update their velocity vector and then their position is adjusted based on the new velocity. The velocity is typically adjusted by a social component (movement towards other particles) and a cognitive component (movement towards the best individual that it has seen so far). The adjustment to the velocity may be modified by a constriction coefficient [Clerc 1999] or an inertia factor [Eberhart and Shi 2000] in an effort to reduce the velocity and to promote convergence. If the new location produced by the move improves the current best, then the current best is updated. If the new location goes beyond the allowable region of the search, the location must be adjusted.

There are number of strategies for the social component. The simplest may be to go towards the best particle in the swarm (Global or Star Topology), but travel towards a logically near particle (Ring Topology) or travel towards a random particle (Random Topology) are also possibile strategies. For a Global Topology, the best particle can be updated synchronously (one update per cycle of all particle moves) or asynchronously (updated before every particle moves).

It should be noted that PSOs have some key differences from the previously mentioned algorithms. First, particles in a serial PSO do not die - they merely move. Second, while particles are initially placed randomly, the movements of the particles are typically governed by strict social rules. This differs from the more random variation operators in GAs. Third, the cognitive component in a particle swarm has no direct analog in a GA.

2.2 Overview of Parallel Evolutionary Systems

Evolutionary Algorithms can be made to operate in parallel in a number of ways. Some methodologies dramatically change the operation of the algorithms from their serial counterparts - others do not. In [Cantú-Paz 1998], the author identifies four general categories of parallel evolutionary algorithms:

1. global single-population master-slave systems
2. single-population fine-grained systems
3. multiple-population coarse-grained systems
4. hierarchical parallel systems

It should noted that these categories were originally specified concerning parallel genetic algorithms, but the terminology is equally applicable to include hill-climbers and PSOs in this context. The system developed in this research is categorized as a type 3 system. Thus, types 1,2, and 4 will be discussed briefly to provide a context for this work, while type 3 will be discussed in greater detail.

2.2.1 Global Single-Population Master-Slave Systems

Master-Slave systems are relatively straightforward in their concept and implementation - a single computer maintains the (typically computationally inexpensive) evolutionary algorithm itself and distributes the evaluation of individuals (generally more computationally expensive) to a number of slave computers whose only job is to assign fitness values to received individuals. In [Bethke 1976], Bethke develops an early implementation of a Master-Slave GA, noting that the speedup of the master-slave system rises as the cost of evaluating individuals increases. This concept is substantiated by more recent results, such as [Punch et al. 1993] and [Grefenstette 1995], which show greater speedup from parallelism as the computational costs begin to dominate the communication overhead incurred by this approach.

In [Grefenstette 1981], Grefenstette explores using asynchronous and synchronous systems. In a synchronous system, the master algorithm produces some number of children, sends them to the slaves to be evaluated, and waits for all of them to return. In an asynchronous system, children are generally created as children return from the slave systems - the master system does not wait for all the children to return. It should be noted that while the theoretical workings of a synchronous system are no different than a serial system, an asynchronous system can have different behavior (imagine the case where a slow slave system returns an individual from a much older generation in the algorithm).

In the ideal case where communication costs are very low, a master-slave system can approach linear speedup.

2.2.2 Single-Population Fine-Grained Systems

Fine-Grained Systems have their individuals mapped to nodes in a way that imposes some sort of topology for the mating of individuals. Often, individuals can only mate with individuals that are logically near to them or are in the same neighborhood. This localized strategy is imposed in order to avoid the cost of global mating operations which may be very expensive for large numbers of processors [Spiessons and Manderick 1991]. There is still a single logical population, but because individuals cannot mate with any individual in the population (as in serial or Master-Slave Evolutionary Algorithms) they are not logically the same as a serial EA. Typically, neighborhoods and geographies are established in such a way as to allow limited intercommunication between groupings of individuals to allow good traits to slowly permeate the entire population.

Often, Fine-Grained Systems attempt to map a single individual to a single processor so that the logical geography of the system is simplified and so that each node can perform the evaluations and changes upon one entity. If the fine-grained algorithm only allows global mating operations, then its behavior is logically equivalent to a serial EA. However, if localized operations are allowed, then the performance will be different. These algorithms are best suited for multi-processor systems, but they can also show improved performance even when implemented on serial systems [Cantú-Paz 1998].

2.2.3 Multiple Population Coarse-Grained Systems

Multiple-Population Coarse-Grained Systems (MPCGSs) operate in parallel several evolutionary algorithms with their own populations. These algorithms may or may not have some means of migration (communicating promising solutions between populations). Further, the subpopulations (also called "demes") may be homogeneous (all subpopulations operate the exact same algorithm) or heterogeneous (subpopulations may operate different algorithms or different types of algorithms). In almost all cases, MPCGSs operate differently than serial evolutionary algorithms.

Multiple Populations without Migration

A straightforward approach to a MPCGS would be simply to distribute the execution of EAs on multiple computers at the same time. When one of the algorithms solves a problem, the system can terminate and report the results. This approach, taken by [Shonkwiler 1993], has been shown to improve performance and intuitively makes sense: if one random algorithm generally solves a problem in a certain amount of time, then multiple algorithms should probabilistically solve the problem in less time.

In one of the earliest works on parallel evolutionary algorithms, [Bossert 1967] suggested multiple population systems in which the populations themselves competed for survival. In other words, poorly fit subpopulations would be eliminated and regenerated randomly during the evolutionary process in order to bolster diversity and reduce the chances of stagnation. In what is more relevant to this work, he also suggested the idea of using migration among populations in order to improve diversity.

Multiple Homogeneous Populations with Migration

One of the major goals of a MPCGS with migration is to promote diversity, which can reduce the chances of premature convergence and improve the quality of solutions. However, MPCGSs with Migration do stand the risk of the so-called "superindividual" problem in which diversity is lost by a highly fit individual being migrated into a population and then being selected to reproduce repeatedly. Effectively, this process can cause diversity to be lost and cause all subpopulations to converge to a suboptimal solution. In this sense, measures should generally be taken to avoid this problem. These measures can take the form of low migration rates, migrating less fit or random individuals (instead of the best), or limiting migration to only times when a population appears to be making too little progress. Some efforts, such as [Pettey and Leuze 1989], have demonstrated that parallel evolutionary algorithms have the theoretical capacity to maintain the efficiency in finding solutions that normal evolutionary systems have. This indicates that the aforementioned problems can be overcome given proper parameter settings.

One approach for the preservation of diversity is to divide the search space among the subpopulations in the parallel system. For example, [Neri and Giordana 1995], the authors use a parallel system for concept learning in which they give different subpopulations a subset of the available training examples. Each subpopulation develops individuals with traits tailored to their specific examples, improving their local fitness. Periodic migration is then used in order to create individuals with a higher global fitness. Another similar approach is taken by [Tsutsui and Fujimoto 1993], in which a parent population spawns child subpopulations that search a subset of the search space. Child populations then periodically report back to the parent if they discover better individuals than those found by the parent.

Many approaches have demonstrated that when subpopulations communicate with one another, they can achieve better results than if they ran independently of one another. In fact, in [Grosso 1985], the author showed that fitness improvement was better when a larger population was divided into subpopulations and run independently with periodic communication. This performance even exceeds an equivalently large population with global mating and variation operators (a serial equivalent). However, if a population was divided into multiple subpopulations without any communication, the performance was actually lower than an equivalent serial genetic algorithm.

In [J.P.Cohoon et al. 1991], the authors use a MPCGS with periodic migration on the K-Partition problem and manage to garner superlinear performance over a similar serial genetic algorithm. In other words, they require fewer function evaluations than would be expected by $evaluations_{serial}/\#processors$. More evidence for using separate subpopulations with periodic migration in MPCGS's appears in [Gordon and Whitley 1993] in which the authors compare a number of single-population systems, MPCGS's, and Fine-Grained systems with one another on a number of difficult minimization problems. They encode global strategy and MPCGS systems for the same four algorithms (additionally, they encode a Fine-Grained System which produces average performance). For their MPCGS's, they use periodic migration (with a static migration rate) and use popular, well-performing Genetic Algorithms to run their subpopulations. Their best results are ultimately provided by a MPCGS.

Multiple Heterogeneous Populations with Migration

The term "heterogeneous" has a number of different meanings in the discipline of computer science, so some care must be taken in describing "Heterogeneous Populations." In this case, heterogeneity generally does not infer anything about the computer hardware running the subpopulations (although this might often be a reasonable assumption in other topics). Heterogeneous subpopulations are different in some algorithmic means - whether that be in the parameter settings of algorithms used, the actual type of algorithms, or in differences in the representation of the individuals in that population. For some problems, these systems might be even better than homogeneous systems since subpopulations can focus on exploration or exploitation, creating an implicit balance between selection pressure and diversity [Alba and Troya 1999].

One of the first works to recognize that the loss of diversity in parallel systems might be controlled by slowing convergence in some of the subpopulations was in [Tanese 1987][Tanese 1989*a*][Tanese 1989*b*], a series of studies on the effects of varying migration rates and migration intervals in parallel systems. For the functions that she tested her Distributed Genetic Algorithm (DGA) on, she found that relatively infrequent migration of a small number of individuals had the best performance. Tanese also proposed that the parameters of different algorithms could be altered to create explorers (promoting diversity) and exploiters (promoting increased selection pressure). In [Belding 1995], the author explored using Tanese's DGA algorithm on the Royal Road class of functions and discovered that migrating relatively large amounts of individuals periodically was most effective. This work demonstrated that Tanese's system was flexible enough to perform well on many different problems, but also that the system parameters must be carefully adjusted in order to maximize performance.

A semi-heterogeneous approach was taken in [Mühlenbein et al. 1991], in which the authors periodically exchange individuals between subpopulations of genetic algorithms. However, if progress in a subpopulation stagnates, the subpopulation applies Hill-Climbing in order to improve its fitness. Their strategy also makes use of neighborhoods of subpopulations in order to make the migration of individuals in the system more gradual (some subpopulations can migrate to multiple neighbors so that high-performance traits can eventually permeate all subpopulations). Their system demonstrates strong performance on testbed of functions.

The ideas of heterogeneity were later implemented successfully in a number of works, although the actual nature of the heterogeneity varies. For example, in [Lin et al. 1994] the authors implement an Injection Island model in which different representation resolutions are used in differing subpopulation. Migration is structured in such a way that individuals can only be "injected" into higher resolution subpopulations from lower resolution subpopulations. This process allows individuals to be steadily refined as they pass through levels of the imposed migration hierarchy. However, it should be noted that this procedure requires different fitness functions for each different resolution level, which increases the complexity of the algorithm development. The Injection Island system has been successfully applied to a number of problems including composite beam design [Punch et al. 1995], single componenet design [Parmee and Vekeria 1997], and flywheel design [Eby et al. 1999].

In an approach similar to the Injection Island, [Hu and Goodman 2002] implemented a Hierarchical Fair Competition Model for Parallel Evolutionary Algorithms (not to be confused with the class of hierarchical parallel evolutionary algorithms discussed later) in which the subpopulations are separated based on fitness levels. Lower fit populations are kept separate from higher fitness bands in order to preserve diversity in the system. Migration is only allowed from lower fitness populations to higher fitness populations when an individual becomes more fit than is allowed by its current subpopulation. This strategy seeks to prevent individuals from being lost from the system that might have beneficial traits, but lower fitness values. The system outperforms a serial genetic programming algorithms and multi-population genetic programming systems on an Analog Circuit Synthesis problem.

In [Adamidis and Petridis 1996], the authors develop an eight heterogeneous subpopulation system for the evolution of the weights for a neural network. Different subpopulations, all genetic algorithms, implement differing variation operators and variation probabilities to encourage or discourage exploration. The topology is global, in that all subpopulations broadcast their best individual at the end of the each epoch to all of the other subpopulations. They compare their results to an equivalent system using homogeneous subpopulations and find that the heterogeneous implementation produces notably improved performance.

Another strategy that takes advantage of heterogeneity in the parameter settings of genetic algorithms is the Gradual Distributed Real-Coded Genetic Algorithm (GDRCGA) approach taken by [Herrera and Lozano 1997] and later expanded in [Herrera and Lozano 2000] and [Alba et al. 2004]. Similarly to [Adamidis and Petridis 1996], the GDRCGA achieves heterogeneity through the modification of the variation rate and operators in its subpopulations. However, it additionally seeks to isolate the subpopulations from each other (in order to prevent premature dominance of one population by an extremely fit individual from another subpopulation) by imposing a cube (and later a hypercube) topology. In other words, each subpopulation only has the ability to migrate individuals in a clearly-defined way designed to promote the maximum benefit from "exploration faces" and "exploitation faces" of the cube. Essentially, this methodology seeks to produce a gradual convergence to a solution rather than a rapid (or premature) convergence.

As has been shown by these studies, heterogeneity can be an effective tool in order to improve performance on some problems. Thus, it is a worthy consideration in the design of parallel evolutionary systems.

2.2.4 Hierarchical Systems

These systems are simply multiple-population systems in which each population is a parallel system in and of itself. A simple example would be a MPCGS which had Master-Slave systems for each subpopulation. Another hierarchical system might be a collection of Fine-Grained systems with periodic migration among them (as in [Gorges-Schleutor 1997]). In general, this is a wide-class of parallel systems which show promise, but also tend to require more hardware or computational overhead than conventional parallel systems.

Chapter 3
The Parallel Evolutionary System

3.1 Introduction

The Parallel Evolutionary System (PES) is an MPCGS working to solve an optimization problem. Unlike with most MPCGS systems, however, the algorithms in the subpopulations need not be homogeneous - multiple different types of algorithms (even those with varying candidate solution representations) may cooperate with each other. The PES contains a centralized buffer (referred to as the "memespace") to facilitate communication between subpopulations. The PES also determines the rules that govern the way the algorithms work: how they communicate, what parameters the algorithms use, and how often they communicate with each other. The piece of software that governs these specifications is the PES Controller, although the terms "PES" and "PES Controller" can be used interchangeably.

3.2 Implementations of each EC

There are four different evolutionary algorithms that the PES may choose from: an Evolutionary Hill-Climber, a Steady-State Genetic Algorithm, a Steady-Generational Genetic Algorithm, and a Particle Swarm Optimizer. The specific details of their implementations follow below.

3.2.1 The Evolutionary Hill-Climber Algorithm

The Evolutionary Hill-Climber (EHC), based loosely on the Random-Mutation Hill-Climber described in [Forrest and Mitchell 1993], first randomly creates a single individual consisting of a chromosome with a number of values corresponding to the problem being solved (see Secction 5). Initial values are chosen from within the allowable range dictated by the problem type. This individual is then evaluated by the fitness function and the fitness is assigned to that individual. In addition to its value, each gene in the chromosome has a corresponding integer bias value, initially set to a value of 1.

On each iteration, a single round of binary tournament selection is utilized to pick two genes in the chromosome. The gene with the greatest bias value is chosen (ties broken randomly) to be mutated. A single Gaussian Mutation is applied to the value, multiplied by the mutation amount δ (determined by the PES controller) and the starting value of the gene itself. This child is then evaluated. If the child has better fitness than the parent, the parent is replaced and the bias value associated with that gene is increased by 1. However, if the child has worse fitness than the parent, it is rejected and the bias value associated with that gene is reduced by 1. However, regardless of how many penalties occur, the bias value can never fall below a value of 1.

3.2.2 The Genetic Algorithms

The Steady-State Genetic Algorithm (SSGA)

First, the SSGA [Davis 1991] generates a population of individuals consisting of a chromosome with a number of values corresponding to the problem being solved. Initial values are determined randomly within the allowable range dictated by the problem type.

These individuals are then evaluated. Then, two binary tournament selections are used to select a first, then a second parent. These two parents are then used to create a child via BLX blend crossover as in [Eshelman and Schaffer 1992], with an α value as described in Section 6.1. Next, a Gaussian Mutation operator is applied to the created individual with a probability μ. The mutation magnitude is modified by a coefficient δ. The child will always replace the worst individual in the population. This process is repeated until the maximum number of allowed function evaluations has expired.

The Steady-Generational Genetic Algorithm (SGGA)

The SGGA, as presented in [Syswerda 1991], operates exactly like the SSGA, except that instead of the offspring always replacing the worst individual in the population, it replaces a random individual (but never the best individual). In general, the SGGA has slightly less selection pressure than the SSGA.

3.2.3 The Particle Swarm Optimizer (PSO)

Based on the results provided in [Carlisle and Dozier 2001], the star topology Particle Swarm Optimizer (GLOB) was chosen. A particle consists of a vector (x) describing current location, the current fitness value (xfit), a vector (v) for current velocity, a vector (p) describing the best location arrived at so far, and the fitness of the p-vector (pfit). The workings of the algorithm (shown in Figure 3.1) are as follows. The PSO randomly initializes the x values according to the constraints of the problem being solved (as described in Section 5). The v values are initially set to 0. Once all the particles have been initialized and evaluated, they are moved through the search space at the rate and direction given by their velocity vector. Velocities are set based on three components: the current velocity, a

Figure 3.1: Pseudocode for the Global Topology Particle Swarm Optimizer

```
initialize function evaluations $f = 0$
initialize particles $S$
initialize $x$ randomly
initialize $v$ to 0
initialize $p$ to starting location
for each particle in $S$ : $xfit = pfit = evaluate(x)$
while (best pfit $<$ desired fitness AND $f <$ some number of function evaluations)
    for each particle in the swarm
        $\phi_{best} = getbest()$
        for each element $i$ in the $x$ vector
            $v_i = K * (v_i + \phi_1 rand()(p_i - x_i) + \phi_2 rand()(\phi_{best} - x_i))$
            $x_i += v_i$
            check $x_i$ to make sure it is within legal boundaries
        evaluate $x$
```

cognitive component ϕ_1 moving towards the best individual ϕ_{best} seen by this particle so far, and a social component ϕ_2 moving towards the best individual in the swarm. The cognitive component is calculated as the difference between the current vector and the best solution seen so far multiplied by the cognitive coefficient ϕ_1. The social component is calculated as the difference between the current vector and the best particle in the swarm multiplied by the social coefficient ϕ_2. This entire velocity is then modified by a constriction coefficient K, as suggested by [Clerc 1999]. The algorithm cycles through the population moving particles and asynchronously updating the best individual. After a particle moves, the algorithm checks to see if it has moved beyond the allowable search space. If so, the particle will be stopped from moving further and be placed at the boundary. If a particle finds a location which is better than its current p-vector, it updates its p-vector and pfitness value.

3.3 Migration in the Parallel System

3.3.1 The Memespace

Each algorithm or subpopulation runs independently of the others, with its only means of interaction being through the indirect exchange of individuals with a centralized buffer or"memespace." This approach is similar to the web-based bank of migrants proposed in [Tang 2002]. The memespace consists of an array of N stored candidate solutions paired with their corresponding fitness values (there is no need for a subpopulation to re-evaluate a candidate solution received from the memespace). All subpopulations (regardless of algorithm type) share the same type of memespace buffer.

During a migration, a subpopulation sends one candidate solution to the memespace and receives one candidate solution in return. When a new candidate solution arrives from a subpopulation to the memespace, it is stored and memespace sends back a random stored candidate solution. If the memespace was previously empty, it will send back the candidate solution it just received. When memespace becomes full, it chooses a random candidate solution and returns it to the subpopulation. Then, memespaces replaces that slot with the CS it received in the migration.

In order for conditions to be consistent for each test run, the meme server array is reset for every run. All the subpopulations begin evolving at the same time and remain synchronized throughout the process. Candidate solutions for different problems are not shared within the same memespace.

3.3.2 Migration Rates and the Size of Memespace

The parameter settings for the size of the memespace and the migration rate are two important parameters for the PES system and intuitively seem related. Larger memespaces have a longer "memory" because they tend to store older candidate solutions longer. This might be advantageous if premature convergence is a problem, since it allows stuck algorithms to migrate to regain lost diversity. However, for some problems re-introducing older solutions might work against convergence by constantly introducing poorly fit individuals.

The migration rate presents similar issues. Very high migration rates tend to be highly disruptive to the population or may exacerbate the superindividual problem discussed in Section 2.2.3 . Very low migration rates may cause the PES system to never migrate and leave the subpopulations in total isolation.

Clearly, there is also the possibility of signifcant interactions between these two parameters. Higher migration rates infer that there will be a greater number of replacements in the memespace and thus reduce the length of its memory. Lower migration rates will conversely increase the memory effect. The interactions between memespace size and migration rate will be explored through experimentation.

3.3.3 Migration Strategies

Each subpopulation in the Parallel Evolutionary System periodically migrates candidate solutions from their population to the memespace. On a given evolutionary cycle, the subpopulation generates a random number between 0 and 1. If the number generated is less than the migration rate [1] (MR) associated with that subpopulation, it sends a candidate solution to the memespace and receives a CS in return. When a particle from a PSO migrates to meme space, only the x-vector and xfit value are mapped to the candidate solution. When a CS is sent to a particle swarm, the CS is mapped to the x-vector of the particle it is replacing. The p-vector and pfitness of that particle is left unchanged (unless the new CS is better the current p-vector and pfitness). The v-vector is set to all 0's initially and is then updated normally from then on. There are a number of possibilities for exactly *which* CS is chosen to migrate from a given subpopulation and for which member of the population is replaced with the CS that is returned. In [Cantú-Paz 2000], the author discusses a number of plausible strategies for migration and their effect on selection pressure intensity, of which four are implemented in this work. All migration strategies considered were global in nature (all subpopulations follow the same migration strategy).

Best Replaces Worst

In the Best Replaces Worst (BRW) strategy, subpopulations always send their best individuals to memespace. When they receive individuals back, they always replace the worst individuals in their population, somewhat similar to the workings of the Steady-State Genetic Algorithm. Of all the migration strategies discussed here, this one provides the greatest selection pressure since memespace will tend to be filled with only above-average individuals which will tend to replace below average individuals in their destination populations.

[1] In the literature, the term "migration rate" is sometimes used to describe the number of individuals exchanged during a migration. In this work, migration rate always refers to the probability on a given cycle that migration will occur.

Best Replaces Random

In the Best Replaces Random (BRR) strategy, as in BRW, subpopulations always send their best individuals to memespace. However, when they receive individuals back, they randomly replace an individual in their population, even if that individual is the best individual in the population. This strategy applies somewhat less selection pressure than BRW since although memespace will tend to be filled with above-average individuals, they may also replace above-average individuals when they migrate into subpopulations.

Random Replaces Worst

In the Random Replaces Worst (RRW) strategy, subpopulations send a random individual in their population to the memespace. When they receive individuals back, they always replace the worst individual in their population. In general, this strategy applies less selection pressure than BRW, since the memespace will tend to be filled with average individuals.

Random Replaces Random

In the Random Replaces Random (RRR) strategy, as in RRW, subpopulations send a random individual to memespace. When they receive an individual in return, they also replace randomly. This strategy provides the least selection pressure of all and has the highest potential for diversity since memespace will be filled with average individuals and those individuals will tend to replace average individuals in the subpopulations.

3.4 Stopping Criteria

Once any subpopulation discovers a candidate solution with fitness levels within the tolerance of the specified problem, it sets a flag indicating to the PES system that it has solved the problem. At the end of the current round in each subpopulation, the PES will terminate and report success and the number of function evaluations required to reach the solution. If the PES exhausts all of its allowed function evaluations, it will terminate and report the maximum number of function evaluations as a result.

3.5 Potential Benefits of this Approach

This buffer strategy provides multiple potential benefits. First, it provides a bank of individuals which preserves diversity in the system longer than the more traditional migration strategies. This can be used as a deterrent to premature convergence, which has been shown to sometimes cause systems to fail [Goldberg et al. 1993]. Second, it is simple and could be applied easily for web-based applications. Third, since the individuals received from memespace are random, migration is generally more gradual than direct subpopulation-to-subpopulation migration.

Chapter 4

The Meta-Parallel Evolutionary System

One of the primary goals of this research is to create a way to discover good parameters for Parallel Evolutionary Systems in a way that reduces the amount of trial-and-error experimentation necessary. The MPES evolves PES configurations, altering parameters to find systems with high success rates and require the fewest average function evaluations.

4.1 The Meta-SSGA

The Meta-SSGA is a SSGA (as described in Section 3.2.2 that encodes into the chromosome the composition of the subpopulations (which algorithm for each of the available subpopulation slots) and various PES parameters (e.g. migration rate(s) and migration rates). Individuals are evaluated by creating a PES system, running it for a number of runs, and then examining the success rates and average required function evaluations for at least one subpopulation to garner a solution. The SSGA uses a population size of 12, $\mu = 0.12$, and $\delta = 0.25$. Algorithm type is represented by integer values and migration rate is represented by a double value. An example encoding for an 8 subpopulation MPES Individual is shown in Figure 4.1.

Differing parameters were held constant for the MPES variants (described below) to gain some insight into which parameters should be included in the general MPES system (MPES IV). These variants are described here and then the results and discussion of experiments relating to each of the variants are in Section 6.3.

Figure 4.1: Representation of an MPES Individual

Heading	*Encoding*	*notes*
Algorithm 0	0	indicates an EHC
Algorithm 1	2	indicates a SGGA
Algorithm 2	3	indicates a PSO
Algorithm 3	1	indicates a SSGA
Algorithm 4	0	indicates an EHC
Algorithm 5	0	indicates an EHC
Algorithm 6	0	indicates an EHC
Algorithm 7	2	indicates a SGGA
Migration Rate	0.53	how often subpopulations migrate to memespace

4.1.1 MPES I: A Global Migration Rate

In this relatively simple approach, the algorithm evolves different configurations of subpopulations and a global migration rate. All possible algorithms (EHC, SSGA, SGGA, and PSO) are available as subpopulations and they all share a single value for migration (all subpopulations, regardless of algorithm type migrate at the same rate). The value for the migration strategy and memesize are passed in as arguments to MPES.

4.1.2 MPES II: Separate Migration Rates

In this variant, the algorithm evolves configurations of subpopulations (as in MPES I), but evolves a different migration rate for each type of algorithm. In other words, PSOs might have a different migration rate than EHCs. The theory behind this variant was that some algorithms may benefit more from migration than other algorithms.

4.1.3 MPES III: Dual-Buffer Strategy

This variant was a more significant departure from the previous MPES systems in that the nature of memespace was changed slightly. As discussed in Section 2, one of the problems with parallel evolutionary algorithms (and, in fact, all genetic algorithms) occurs when one highly fit individual overtakes all populations and causes premature, suboptimal convergence. This is often addressed by a change in topology which encourages a more gradual exchange of individuals. In the PES system, the memespace is already indirect and more gradual than direct migration. However, in MPES III an even more gradual approach was taken.

Instead of using a single buffer, MPES III uses two memespaces which periodically exchange individuals between one another. The first four algorithms in the PES system communicate exclusively with the first memespace and the remaining algorithms all communicate with the second memespace. This essentially creates two groups of four subpopulations working together. The global migration rate is set (as in MPES I), but an additional migration rate called the swap rate determines how often the two buffers exchange individuals between each other. On each local evolutionary cycle in the PES, if a random number between 0 and 1 is less than the swap rate, each buffer sends its best individual to the other buffer. This could occur multiple times during one global cycle (in other words, each subpopulation on each iteration has a chance to spawn a swap between the buffers). Individual replacement in each memespace occurs in exactly the same manner as described in Section 3.3. An example individual for MPES III is shown in Figure 4.2.

4.1.4 MPES IV: Evolving all PES Parameters

Using the information gained from the previous experiments, MPES IV attempts to reduce the amount of *a priori* knowledge required to identify good parameters for the PES system by evolving nearly all of them. It has the option of a dual-buffer strategy. MPES

Figure 4.2: Representation of an MPES III Individual

Heading	*Encoding*	*notes*
Algorithm 0	0	migrates to first memespace
Algorithm 1	2	migrates to first memespace
Algorithm 2	3	migrates to first memespace
Algorithm 3	1	migrates to first memespace
Algorithm 4	0	migrates to second memespace
Algorithm 5	0	migrates to second memespace
Algorithm 6	0	migrates to second memespace
Algorithm 7	2	migrates to second memespace
Migration Rate	0.53	how often subpopulations migrate
Swap Rate	0.2	how often the memespaces exchange individuals

IV evolves the configuration of subpopulations, the migration strategy, global migration rate, the swap rate, and the number of subpopulations which communicate with the first memespace (if this number is 0 or greater than the total number of subpopulations, then obviously only one buffer will be used). The goal of this variant is to identify PES systems that will have comparable performance to those discovered through extensive experimentation. The advantage would be that this system would require less effort on the part of the configuration designer since it would simply require running the MPES IV system. Even if the system is not able to identify the absolute global optimum configuration, if it can relatively quickly identify well-performing configurations it can still be a valuable tool.

Chapter 5

The Test Suite

5.1 Function Minimizers

In order to evaluate and compare the algorithms, they were tasked to minimize five of the benchmark functions utilized by the authors in [Eberhart and Shi 2000]. Specifically, they are the Schaffer F6 function, the Sphere (De Jong F1) function, the Rosenbrock function, the Rastrigrin function, and the Griewank function. All of these functions are nonlinear and feature a complex search space.

5.2 Schaffer F6

The F6 Function (Equation 5.1) receives two inputs - an x and a y value. These values are constrained to the range of [-100,100]. The global minimum for this function is $f(x) = 0$.

$$F6(x, y) = 0.5 + \frac{(\sin^2(\sqrt{x^2 + y^2}) - 0.5)}{(1.0 + 0.001(x^2 + y^2))^2} \tag{5.1}$$

5.3 Sphere (de Jong F1)

The F1 Function (Equation 5.2) receives an array of n=30 inputs, $x_1 - x_{30}$. The allowable range for the elements was [-30,30]. The global minimum for this function is $f(x) = 0$.

$$F1(x1...x30) = \sum_{i=1}^{n} x_i^2 \tag{5.2}$$

5.4 Rosenbrock Function

The Rosenbrock Function (Equation 5.3) receives an array of n = 30 inputs, $x_1 - x_{30}$. The allowable range for these values is [-30,30]. The global minimum for this function is $f(x) = 0$.

$$Rosenbrock(x1...x30) = \sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2) + (x_i - 1)^2) \tag{5.3}$$

5.5 Rastrigin Function

The Rastrigin Function (Equation 5.4) receives an array of n = 30 inputs, $x_1 - x_{30}$. The allowable range for these values is [-5.12,5.12]. The global minimum for this function is $f(x) = 0$.

$$Rastrigin(x1...x30) = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) \tag{5.4}$$

5.6 Griewank Function

The Griewank Function (Equation 5.5) receives an array of n = 30 inputs, $x_1 - x_{30}$. The allowable range for these values is [-600,600]. The global minimum for this function is $f(x) = 0$.

$$Griewank(x1...x30) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{5.5}$$

5.7 Easom Function

The Easom Function (Equation 5.6) receives two inputs, x and y. These values are constrained to the range of [-100,100]. The function has one global minimum with a value of $f(x, y) = -1$ when $x = \pi$ and $y = \pi$.

$$Easom(x, y) = -\cos(x)\cos(y) * \exp(-(x-\pi)^2 - (y-\pi)^2) \tag{5.6}$$

Chapter 6

Experiments and Discussion

This section describes the experiments done relating the MPES system. They are presented in roughly chronological order, as many of the later results are based on ideas gained from earlier results.

Throughout this chapter the abbreviation "SR" refers to the Success Rate, defined as the percentage of runs in which at least one subpopulation of the parallel system achieved a candidate solution with fitness within the tolerance specified in Section 5. Additionally, "AFE" stands for Average Function Evaluations, which is the average number of evaluations that each subpopulation had to execute before a solution was found. Since this is a parallel system, it should be noted that to calculate the system-wide number of function evaluations would require multiplying the AFE by the number of subpopulations in the PES system. The term "best performance" always indicates the run with the greatest SR, unless specifically stated otherwise. If the SR is the same for two sets, then the algorithm with the lowest AVE is considered the best. If both SR and AVE are the same, then the algorithm with the lowest average raw fitness is chosen (the raw fitness is not reported here, it is only relevant for the selection of the good parameters for the individual algorithms).

6.1 Parameters for the Individual Algorithms

Before running experiments with the PES with migration, the experiments were run on the individual algorithms to find good parameter settings for each algorithm on each problem. In this case, an 8-subpopulation PES system with a migration rate of 0 was used and each PES system was run 100 times for each parameter configuration. The goal was not necessarily to find optimal parameter settings, but rather to identify those with reasonable performance to use in the other experiments. It should be noted that several of the SRs are very low, even for the best settings. This is because most of the problems in the test suite are difficult to solve in 2000 iterations to the level of accuracy desired. The performance of these systems improves later when migration operators are introduced. A description of the parameter configurations experimented with and the best results for each algorithm are reported below.

6.1.1 Optimizing the Evolutionary Hill-Climber

The single parameter of interest for the EHC is the δ value or mutation amount. The δ-values were taken from the set $\delta = [0.03, 0.06, 0.12, 0.25, 0.33, 0.5, 0.66, 0.75, 0.875, 1.0, 1.25, 1.33, 1.5, 1.66, 1.75, 2.0, 3.0]$. The best performing δ value for each problem is reported in Table 6.1, along with the AFE and the SR values corresponding to those runs. In general, a good delta rate seems to be near 1.0 for all of the problems, which seems reasonable since it allows the EHC to find solutions that are far away from the current candidate solution. A smaller delta would likely make the EHC too susceptible to getting stuck at a local minima. In general, the EHCs perform well on the Sphere, Rastrigin, and Griewank functions (garnering a 1.00 SR), but seems less effective on the Schaffer, Rosenbrock, and Easom functions. The poor performance on the Easom function particularly makes sense given that in order to find much improvement in fitness, the individual must already be very close to the global minima. Algorithms using populations greater than one should perform better on this function.

Table 6.1: Best Parameter Settings and Performance for EHC

problem	δ	*AFE*	*SR*
Schaffer	0.875	219.93	0.95
Sphere	1.000	980.20	1.00
Rosenbrock	1.000	1572.66	0.85
Rastrigin	0.875	225.77	1.00
Griewank	1.000	1280.81	1.00
Easom	1.330	293.50	0.89

In general, the anticipation is that the EHC algorithms will have the role of exploiters in systems with migration. When they fail, it is typically due to their inability to break away from local minima (even with a relatively high delta rate). Migration should provide them with a mechanism to avoid this pitfall.

6.1.2 Optimizing the Genetic Algorithms

Both the SSGA and the SGGA were tested on the same variety of parameter settings. Crossover was always performed with a probability of 1.0, and both a standard uniform crossover and a BLX-α crossover with α varying from the set [0.25,0.5,1.0] were tested. For all problems and for both algorithms, some variant of BLX-crossover was the best performer. Additionally, for each algorithm the population size was varied from the set [3,6,12,25,50,100], the mutation rate μ was varied from the set [0.03,0.06,0.12,0.25], and finally the mutation amount δ was varied from the set [0.03,0.06,0.12,0.25]. The best performing iparameters for the SSGA are reported in Table 6.2 and the best performing parameters for the SGGA are reported in Table 6.3. In the absence of migration, the SSGAs and SGGAs perform fairly poorly on all problems excepting the Rastrigin and Easom Function. In fact, it never successfully solves the Sphere, Rosenbrock, or Griewank problems. In general, for both of the GAs, low delta rates are common. In general, the SSGA seems

Table 6.2: Best Parameter Settings and Performance for SSGA

problem	*pop.size*	μ	δ	α	*AFE*	*SR*
Schaffer	25	0.25	0.03	1.0	1827.32	0.31
Sphere	12	0.03	0.03	1.0	2000.00	0
Rosenbrock	6	0.06	0.03	0.5	2000.00	0
Rastrigin	3	0.06	0.06	0.25	952.32	1.00
Griewank	12	0.03	0.03	1.0	2000.00	0
Easom	6	0.25	0.03	1.0	114.86	1.0

Table 6.3: Best Parameter Settings and Performance for SGGA

problem	*pop.size*	μ	δ	α	*AFE*	*SR*
Schaffer	12	0.06	0.03	0.5	1756.30	0.35
Sphere	3	0.12	0.03	0.5	2000.00	0
Rosenbrock	3	0.03	0.03	0.25	2000.00	0
Rastrigin	3	0.06	0.06	1.0	1199.66	1.00
Griewank	3	0.12	0.03	0.5	2000.00	0
Easom	6	0.25	0.03	0.25	181.28	0.99

to perform better with slightly larger population sizes than the SGGA. This makes sense, because the SSGA has slightly higher selection pressure than the SGGA and would require a larger population to maintain diversity.

In general, the GAs will likely be a balance between exploration and exploitation in heterogeneous systems.

6.1.3 Optimizing the Particle Swarm Optimizer

The Parameters for the PSOs were strongly influenced based on the suggested parameter settings in [Carlisle and Dozier 2001]. Specifically, for all problems the swarm size was set to 30, the constriction coefficient was set to 0.73, the star or global topology was used, and the updates to the best particle were performed asynchronously. However, the best settings for the social and cognitive components (ϕ_1 and ϕ_2 respectively) differed slightly in some cases than those reported by [Carlisle and Dozier 2001]. For all trials, we used

Table 6.4: Best Parameter Settings and Performance for the PSO

problem	ϕ_1	ϕ_2	AFE	SR
Schaffer	2.9	1.2	1985.63	0.07
Sphere	2.9	1.2	2000.00	0
Rosenbrock	3.1	1.0	2000.00	0
Rastrigin	2.5	1.6	1995.24	0.05
Griewank	2.9	1.2	2000.00	0
Easom	3.0	1.1	383.04	1.00

$\phi = \phi_1 + \phi_2 = 4.1$ and ϕ_1 varied from the range 2.0 - 3.6 in increments of 0.1. The best settings for each problem are reported in Table 6.4.

In general, the PSOs have trouble solving all of the problems, except for the Easom Function. In general, we consider the PSOs with these settings strong explorers which may prove beneficial in systems with migration in order to constantly replenish diversity in the system.

6.2 Homogeneous Algorithms with Migration

Once reasonable parameters were obtained for each individual algorithm, the PES system was tested using the best settings from above, plus migration. This was done to see if the use of the memespace as a migration mechanism was effective and to provide a basis for comparison with heterogeneous systems and the later results provided by MPES.

In these experiments, an 8-subpopulation PES system with a global migration rate (meaning that all subpopulations have the same probability of migrating) taken from the set [0.0025, 0.005, 0.0075, 0.01, 0.02, 0.03, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9] was used and each PES system was run 100 times for each parameter configuration. For each migration rate, we tested PES with a memesize taken from the set [2,4,8,16,32].

Table 6.5: Best Performance for the Homogeneous Hill-Climbers

problem	*MR*	*MS*	*memesize*	*AFE*	*SR*	*relative*
Schaffer	**0.2**	**RRR**	**2**	**139.40**	**0.96**	**+0.13**
Sphere	**0.01**	**BRW**	**32**	**433.39**	**1.00**	**+0.56**
Rosenbrock	**0.35**	**BRW**	**2**	**1226.59**	**0.96**	**+0.22**
Rastrigin	**0.03**	**BRR**	**32**	**109.92**	**1.00**	**+0.51**
Griewank	**0.45**	**BRR**	**2**	**1162.70**	**1.00**	**+0.09**
Easom	0.4	RRR	8	181.91	0.97	+0.38

Additionally, for each of the above configurations, we tested the migration rate plus memespace pair on each of the migration strategies described in Section 3.3.3, which are BRW, BRR, RRW, and RRR. A description of the parameter configurations experimented with and the best results for each algorithm are reported below. If the parameter settings indicate a "*" this means that the algorithm had a SR of 0. The "relative" column compares the performance of the configuration with migration to the corresponding configuration without migration (e.g. $\frac{AFE_{noMR}-AFE_{MR}}{AFE_{noMR}}$). If a given system configuration is the best among all the homogeneous algorithms, it is bolded.

Although all four migration strategies were tested, it should be noted that there is no actual difference in the system performance for an EHC-only PES. This is because, in a single member population, there is only one choice of individual to send to memespace and likewise there is only one individual to replace when an individual is returned. Since the migration strategy does not affect which individual memespace sends back (always random), it has no effect.

As can be seen in Table 6.5, the EHCs configuration performs the best on almost all of the problems (excepting the Easom Function, in which the EHC seems to still suffer from the problems discussed earlier). Even relatively low migration rates provide increases in performance on the Sphere and Rastrigin functions. Migration for the EHCs seems to

Table 6.6: Best Performance for the Homogeneous SSGAs

problem	*MR*	*MS*	*memesize*	*AFE*	*SR*	*relative*
Schaffer	0.01	BRR	8	1591.27	0.71	+0.13
Sphere	0.9	BRW	2	1557.95	0.95	+0.22
Rosenbrock	0.45	BRW	16	1661.16	0.57	+0.17
Rastrigin	0.9	BRR	2	206.99	100	+0.78
Griewank	0.9	BRR	4	1845.39	0.59	+0.08
Easom	0.7	BRR	2	45.43	1.00	+0.60

be beneficial in terms of the algorithms ability to sometimes gain older solutions, in effect back-tracking, through migration. This type of backtracking is typically not available for the EHC. The best performing settings trend towards medium migration rates for small memespaces (note Schaffer, Rosenbrock, and Griewank) and towards small migration rates for large memespaces (note Sphere and Rastrigin).

For the Homogeneous SSGAs with migration (see Table 6.6), the systems outperform the corresponding systems without migration. The best-performing systems tend to be ones with medium to high migration rates. The high migration rates tend to be paired with relatively small memespaces, which is actually quite different than was seen with the EHCs. This suggests that the SSGA needs a steady influx of individuals from other populations and that the memory of the memespace should be short-lived for the Sphere, Rastrigin, Griewank, and Easom problems. For the Schaffer problem, migration is infrequent and memespace is medium. For the Rosenbrock, migration is medium and memespace is fairly large. In general, the migration strategies that perform best seem to be BRW and BRR (in fact, the other strategies never produce the best performance for either of the Genetic Algorithms).

Many of the trends noted for the SSGA also appear in the SGGA (see Table 6.7, although the SGGA generally performs worse than the SSGA (particularly on the Sphere and

Table 6.7: Best Performance for the Homogeneous SGGAs

problem	*MR*	*MS*	*memesize*	*AFE*	*SR*	*relative*
Schaffer	0.01	1	32	1567.8	0.63	+0.10
Sphere	*	*	*	2000.00	0	0
Rosenbrock	0.5	0	4	1584.9	0.64	+0.21
Rastrigin	0.9	0	2	256.69	1.00	+0.79
Griewank	*	*	*	2000.00	0	0
Easom	**0.7**	**0**	**2**	**41.13**	**1.00**	**+0.77**

Table 6.8: Best Performance for the Homogeneous PSOs

problem	*MR*	*MS*	*memesize*	*AFE*	*SR*	*relative*
Schaffer	0.02	BRR	2	1598.87	0.68	+0.19
Sphere	0.5	RRW	4	1091.65	1.00	+0.45
Rosenbrock	0.3	RRW	4	1323.04	0.95	+0.34
Rastrigin	0.9	RRW	2	321.16	1.00	+0.84
Griewank	0.6	RRW	2	1163.68	1.00	+0.42
Easom	0.25	BRR	16	121.77	1.00	+0.68

Griewank problems). We again note that low migration rates tend to be paired with large memespaces and that larger migration rates tend to be paired with smaller memespaces. All of the migration strategies are either BRW or BRR.

The PSOs provide success rates that are comparable to the EHCs for most part, although have large AFE values. The PSOs have difficulty solving the Schaffer function and demonstrate performance similar to the Genetic Algorithms. Interestingly, the migration strategy RRW proves the best for the PSO in most cases, indicating that the PSO benefits from a slightly lower pressure migration strategy. In general, the PSOs seem to gain the most of all the algorithms from migration in terms of relative performance.

6.2.1 General Observations

There are several key observations to be gained from these experiments. First, the hardest problems in this set seem to be the Sphere, Rosenbrock, and Griewank. In fact, no PES system managed to achieve a SR of 1.00 on the Rosenbrock Function. The AVE values for these algorithms are notably higher than the other problems as well. Second, migration improves performance relative to non-migrating systems for nearly all problems for all algorithms. The only exception to this is the fact that the SGGA still fails to solve the Sphere and Griewank problems, so no data on relative performance was gained. It should be noted that migration never reduced the performance in terms of function evaluations, although migration does incur a cost in terms of communication. Higher migration rates would be undesirable in systems where the cost of evaluating a function was much cheaper than the cost to communicate between memespace and the subpopulation. However, if function evaluations are relatively expensive, then the cost of communication seems to be worthwhile. A third observation is that the migration strategy RRR (the one with the highest likelihood of promoting diversity) never appears in the best settings for any algorithm.

6.3 MPES

For MPES, the metrics for performance are the same as those reported above. However, now the system has the possibility of having heterogeneous subalgorithms. For MPES I - III, the MetaSSGA is given 500 evaluations to evolve. In MPES IV, the system is given 2000 evaluations since it is evolving a larger number of parameters. Each subpopulation is given 2000 function evaluations. If a configuration is a homogeneous set of EHCs, then the migration strategy is listed as “NA” to avoid distracting the reader from the fact that the migration strategy in such systems is irrelevant.

Table 6.9: Best Performing Configurations for MPES I

problem	*configuration*	*MR*	*memesize*	*MS*	*AFE*	*SR*	*relative*	*sig*
Schaffer	00000001	0.16	2	RRW	147.07	1.00	+0.23	no
Sphere	00000000	0.005	32	NA	462.37	1.00	-0.07	no
Rosenbrock	00000000	0.57	2	NA	1224.37	0.99	+0.00	no
Rastrigin	00000000	0.036	32	NA	105.34	1.00	+0.04	no
Griewank	00000013	0.75	2	BRW	655.56	1.00	+0.44	yes
Easom	01222222	0.75	2	BRW	36.1	1.00	+0.12	yes

6.3.1 MPES I: A Global Migration Rate

In the MPES I system, the system evolved the subpopulation configurations [1] and the global migration rate. The system was not allowed to evolve the migration strategy or memesize. This was done in order to see the effects on performance that these two parameters had on the MPES system. MPES I was tested with memesizes and migration strategies varying as described in Section 6.2. All algorithms are allowed and the MR is allowed to vary between [0.005, 0.75], since in the previous experiments with migration, no best configuration had migration rates outside of this range. The relative performance for all configurations is given with respect to the best performing homogeneous configuration with migration. The "sig" column indicates if the difference in performance is statistically significant, as determined by a pair-wise t-test.

As Table 6.9 shows, MPES I generally identifies PES systems that perform about as well or better than the best homogeneous settings (in three cases, the homogeneous settings are the best). The largest improvement discovered was for the Griewank Function, in which integrating a SSGA and PSO into a population of six EHC algorithms provides a sizeable

[1] In all configurations, the code 0 refers to an EHC, code 1 refers to a SSGA, code 2 refers to a SGGA, and code 3 refers to a PSO. For example, "00000123" would represent a system with 5 EHCs, 1 SSGA, 1 SGGA, and 1 PSO. In cases where the order of the chromosome matters (e.g. MPES III), the order presented will be the actual order in which the algorithms appear on the chromosome.

increase in performance. With respect to the sizing of memespace, the best results follow a similar pattern as they did in the homogeneous cases (in fact, the best performing size for memespace is the same for both).

6.3.2 MPES II: Separate Migration Rates for Each Algorithm Type

It should be noted that in Table 6.10, only a single migration rate appears, when in the actual encoding four migration rates were evolved. The reason for this is because in every single case, all applicable migration rates evolved to the same value (the migration rates are irrelevant for those algorithms which were not present in any subpopulations). This result was somewhat surprising, but explains why the results are generally the same as the best results found by MPES I. The exception to this is the performance by MPES II on the Schaffer problem was mildly better (and the difference is statistically significant). Additionally, the best performance on the Schaffer Problem in this case is a homogeneous solution instead of the heterogeneous solution discovered by MPES I. The results found by MPES II provide some useful information. First, that a simplistic global migration rate seems to be a satisfactory approach for PES systems and provides the added advantage of reducing the number of parameters to evolve. Second, since MPES II found settings with performance similar to or better than the strictly homogeneous bests, it shows that the MPES systems can fairly consistently identify good settings for PES.

6.3.3 MPES III: Dual-Buffer Strategy

The primary goal of the Dual-Buffer Strategy is to make the migration between population more gradual since the only way for individuals to migrate between the two sub-groups is through the swap operation. In Table 6.11, the best performing configurations for MPES

Table 6.10: Best Performing Configurations for MPES II

prob.	*configuration*	*MR*	*memesize*	*MS*	*AFE*	*SR*	*relative*	*sig*
Sch.	00000000	0.75	2	NA	107.51	1.00	+0.27	yes
Sph.	00000000	0.005	32	NA	432.19	1.00	+0.07	no
Ros.	00000000	0.75	2	NA	1194.94	0.96	+0.02	no
Ras.	00000000	0.025	32	NA	104.65	1.00	+0.01	no
Gri.	00000013	0.75	2	BRW	657.6	1.00	-0.00	no
Eas.	00122222	0.75	2	BRW	37.06	1.00	-0.03	no

Table 6.11: Best Performing Configurations for MPES III

prob.	*configuration*	*MR*	*Swap*	*memesize*	*MS*	*AFE*	*SR*	*relative*	*sig*
Sch.	0002 0002	0.4	0.005	4	RRW	204.59	1.00	-0.39	yes
Sph.	0000 0000	0.75	0.75	4	NA	381.06	1.00	+0.18	yes
Ros.	0000 0000	0.75	0.75	8	NA	902.05	0.86	+0.26	yes
Ras.	0001 0000	0.75	0.75	4	BRW	92.11	1.00	+0.13	yes
Gri.	0000 0000	0.75	0.75	4	NA	412.96	1.00	+0.37	yes
Eas.	1222 0222	0.75	0.75	2	BRW	35.25	1.00	+0.02	no

III are shown. In the configuration column, the first four subpopulations constitute the first group and always migrate to the first memespace buffer. Likewise, the second four subpopulations always migrate to the second memespace buffer. The swap column indicates the probability on any given local cycle that the two buffers will exchange individuals, as described in 4. The memesize column has the same meaning as before, but note that this is the size of *each* of the two buffers, so the total buffer space would be twice the memesize value.

The performance of MPES III is mixed on the varying algorithms. It performs exceptionally well on Sphere, Rastrigin, and Griewank problems. Specifically, it garners the best performance of any of the PES configurations discovered so far on those problems and the results are statistically different. While the AFE performance on the Rosenbrock is decidedly lower, the success rate falls about 0.10 over previous best configurations. MPES

III has difficulty finding a fast configuration for the Schaffer problem, but it still maintains a success rate of 1.00. In general, the memesizes are smaller (even accounting for the fact that there are two of them) which indicates that for the problems that require gradual convergence, the double-buffer strategy works better than one large memespace. It should be noted that this is the case even with the relatively high swap rates which performed the best. The migration strategies in this case are, once again, all BRW and RRW strategies indicating that increased selection pressure from these strategies is most effective.

6.3.4 MPES IV: Evolving all PES Parameters

In order to fulfill the goal of minimal effort on the part of the PES designer, MPES was tested with encoding all the major PES parameters. The configuration can be changed as in the previous MPESs. The value of cutpoint is an integer from the range [0-8], which determines which subpopulations communicate with which memespace buffer. Note, than unlike in MPES III, in this way configurations are possible in which the subpopulations are not dividedly evenly between the two groups (e.g. a cutpoint of 5 would indicate that 5 subpopulations migrate to the first memespace and 3 subpopulations migrate to the second memespace). When a dual-buffer strategy is represented in this section, the configuration shows a space where the cutpoint takes place. The meanings of migration rate and swap rate are unchanged and allowed to vary as in MPES III. The memesize is now being evolved, with its value taken from the range [2-32]. The migration strategy is allowed to vary from BRW, BRR, and RRW (RRR was removed for poor past performance). The best performing configurations discovered by MPES IV are shown in Table 6.12.

Table 6.12: Best Performing Configurations for MPES IV

prob.	*configuration*	*CP*	*MR*	*Swap*	*memesize*	*MS*
Sch.	00000000	8	0.37	NA	3	NA
Sph.	0000 0000	4	0.75	0.75	7	NA
Ros.	00000000	8	0.374	NA	2	NA
Ras.	300000 00	6	0.75	0.75	3	BRR
Gri.	00000 000	5	0.75	0.75	8	BRW
Eas.	02 222112	2	0.75	0.7	4	BRW

As seen in Table 6.12, the best performing configurations are similar to the best configurations found by previous MPES's, but did not require nearly as exhaustive runs in order to find them. The dual-buffer strategy appears in two-thirds of the configurations and it is interesting that the subpopulations are not always divided into equal groups. Homogeneous EHCs appear to be the best for solving the Schaffer, Sphere, Rosenbrock, and Griewank problems. Heterogeneous solutions perform the best on the Rastrigin and Easom problems. The actual performance measures for those configurations are presented in Table 6.13. The AVE and SR columns are unchanged in meaning. The cycles column indicate (approximately) how many MPES cycles were required to find these settings. The "rel" column indicates the relative performance of the MPES IV solution to the best solution seen so far ("best" indicates which MPES system found that best solution for the convenience of the reader). The "sig" column indicates whether or not the performance was statistically different.

In general, the performance of MPES IV is fairly close to the best settings found by the more exhaustive methods in the previous MPES trials. No MPES IV trial required the full 2000 runs in order to find its best settings. In fact, other than the Schaffer Problem, the settings were found in less than 1000 iterations. For the Rosenbrock, Rastrigin, and Easom problems, MPES IV performance is indistinct from the bests found elsewhere. For

Table 6.13: Performance for Best Settings Found by MPES IV

prob.	*AVE*	*SR*	*cycles*	*rel*	*best*	*sig*
Sch.	126.91	1.00	1180	-0.18	MPESII	yes
Sph.	410.65	1.00	440	-0.08	MPESIII	yes
Ros.	1239.42	0.97	980	-0.01	MPESI	no
Ras.	90.55	1.00	660	+0.02	MPES III	no
Gri.	451.7	1.00	640	-0.09	MPESIII	yes
Eas.	35.01	1.00	540	+0.01	MPESIII	no

the Sphere and Griewank problems, the performance is only slightly worse than the bests (less than 0.10 AVE off). All of the solutions provided by MPES IV have an SR of 1.00, except for the Rosenbrock Function which garners an SR of 0.97. It should be noted that even in the worst case (the Schaffer Function), the performance is still within 20 AVE of the best found.

6.4 Notes on Performance

In order to run MPES IV on the entire test suite on 3.0 Ghz Intel Machines required approximately 24 hours. The author recognizes that in order to apply MPES to systems with considerably longer function evaluation times, some measures might need to be taken in order to reduce runtime. There are several ways that this could be accomplished. First, MPES could simply be given fewer iterations. This will likely reduce the quality of PES systems that it finds, but that may be acceptable for many applications. Second, the number of runs given to test each PES configuration in order to assign fitness to them could be reduced considerably. For example, in this work each configuration was run 100 times, but this number could be reduced considerably at the risk of diminishing statistical significance. Third, the sub-populations could be given fewer iterations which might be acceptable for some problem types. Fourth, MPES could be programmed to stop as soon as it found a PES system that had an SR of 1.00 (in many applications, this might be the only necessary outcome).

Chapter 7
Conclusion

In accordance with the goals set out in Section 1.1, a Parallel Evolutionary System was developed. These Parallel Evolutionary Systems ultimately made use of one or more centralized buffers for the exchange of individuals in order to perform gradual, indirect communication between potentially heterogeneous members of the subpopulation. Several algorithms to govern the subpopulations of the system were implemented and good parameters for each of those algorithms were discovered. Experiments were performed in order to discover migration rates, appropriate sizing for memespace, and high-performance migration strategies.

Several Meta-Parallel Evolutionary Systems were developed and extensively tested on a suite of difficult optimization problems to consistently develop high-performance Parallel Evolutionary Systems. The MPES optimized configurations of subpopulations and general PES parameters. Ultimately, a fairly general version of MPES was created based on the knowledge gained from previous experiments. The author believes that this version of MPES can consistently be used in order to efficiently develop high performance PESs for a wide variety of difficult problems with considerably less effort and greater success for the PES designer than was previously required by ad hoc methodologies.

Bibliography

Adamidis, P. and Petridis, V. [1996], Co-operating Populations with Different Evolution Behaviors, *in* 'Proceedings of the 1996 IEEE International Conference on Evolutionary Computation', pp. 188–191.

Alba, E., Luna, F., Nebro, A. J. and Troya, J. M. [2004], 'Parallel heterogeneous genetic algorithms for continuous optimization', *Parallel Comput.* **30**(5-6), 699–719.

Alba, E. and Troya, J. M. [1999], 'A survey of parallel distributed genetic algorithms', *Complexity* **4**(4), 31–52.

Bäck, T., Hammel, U. and Schwefel, H. [1997], 'Evolutionary Computation: Comments on the History and Current State', *IEEE Transactions on Evolutionary Computation* **1**(1).

Belding, T. [1995], The Distributed Genetic Algorithm Revisited, *in* 'Proceedings of the Sixth International Conference on Genetic Algorithms', pp. 114–121.

Bethke, A. [1976], Comparison of Genetic Algorithms and Gradient-Based Optimizers on Parallel Processors: Efficiency of Use of Processing Capacity, Technical Report 197, University of Michigan.

Bossert, W. [1967], 'Mathematical Optimization: Are There Abstract Limits on Natural Selection?', *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution* pp. 35–46.

Cantú-Paz, E. [1998], 'A Survey of Parallel Genetic Algorithms', *Calculateurs Parallels, Reseaux et Systems Repartis* **10**(2), 141–171.

Cantú-Paz, E. [2000], *Efficient and Accurate Parallel Genetic Algorithms*, Kulwer Academic Publishers, Boston.

Carlisle, A. and Dozier, G. [2001], An Off-The-Shelf PSO, *in* 'Proceedings of the 2001 Workshop on Particle Swarm Optimization', pp. 1–6.

Clerc, M. [1999], The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization, *in* 'Proceedings of ICEC', pp. 1951–1957.

Davis, L. [1991], *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

DeJong, K. and Spears, W. M. [1993], On the State of Evolutionary Computation, *in* 'The Proceedings of the International Conference on Genetic Algorithms', pp. 618–623.

Dozier, G., Britt, W., SanSoucie, M., Hull, P., Tinker, M., Unger, R., Bancroft, S., Moeller, T. and Rooney, D. [2006], 'Evolving High-Performance Evolutionary Computations for Space Vehicle Design', *IEEE Congress on Evolutionary Computation* pp. 2201 – 2207.

Eberhart, R. and Shi, Y. [2000], Comparing inertia weights and constriction factors in particle swarm optimization, *in* 'Congress on Evolutionary Computation, vol. 1', pp. 84–88.

Eby, D., Averill, R., Goodman, E. and Punch, W. [1999], 'Optimal Design of Flywheels Using an Injection Island Genetic Algorithm', *Artificial Intelligence in Engineering Design, Analysis and Manufacturing* **13**, 389–402.

Eshelman, L. J. and Schaffer, J. D. [1992], Real-coded genetic algorithms and interval-schemata., *in* 'FOGA', pp. 187–202.

Fogel, D. [1995], *Toward a New Philosophy of Machine Intelligence*, IEEE Press.

Forrest, S. and Mitchell, M. [1993], Relative building-block fitness and the building-block hypothesis, *in* L. D. Whitley, ed., 'Foundations of Genetic Algorithms 2', Morgan Kaufmann, San Mateo, CA, pp. 109–126.
URL: *citeseer.ist.psu.edu/forrest93relative.html*

Golberg, D. [1989], *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

Goldberg, D., Deb, K. and Thierens, D. [1993], 'Toward a Better Understanding of Mixing in Genetic Algorithms', *Journal of the Society of Instrument and Control Engineers* **32**, 10–16.

Gordon, V. S. and Whitley, D. [1993], Serial and Parallel Genetic Algorithms as Function Optimizers, *in* 'Proceedings of the Fifth International Conference on Genetic Algorithms', pp. 177 – 183.

Gorges-Schleutor, M. [1997], ASPARAGOS96 and the Traveling Salesman Problem, *in* 'Proceedings of the Fourth International Conference on Evolutionary Computation', pp. 171–174.

Grefenstette, J. [1981], Parallel Adaptive Algorithms for Function Optimization, Technical Report CS-81-19, Vanderbilt University.

Grefenstette, J. [1986], 'Optimization of Control Parameters for Genetic Algorithms', *IEEE Transactions on Systems, Man, and Cybernetics* **1**, 122–128.

Grefenstette, J. [1995], Robot Learning with Parallel Genetic Algorithms on Networked Computers, *in* 'Proceedings of the 1995 Summer Computer Simulation Conference', pp. 352–357.

Grosso, P. B. [1985], Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model, PhD thesis, University of Michigan.

Herrera, F. and Lozano, M. [1997], Heterogeneous Distributed Genetic Algorithms Based on the Crossover Operator, *in* 'Genetic Algorithms in Engineering Systems: Innovations and Applications', pp. 203–208.

Herrera, F. and Lozano, M. [2000], 'Gradual Distributed Real-Coded Genetic Algorithms', *IEEE Transactions on Evolutionary Computation* **4**, 43–63.

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

Hu, J. J. and Goodman, E. [2002], The Hierarchical Fair Competition (HFC) Model for Parallel Evolutionary Algorithms, *in* 'Proceedings of the 2002 Congress on Evolutionary Computation', pp. 49–54.

J.P.Cohoon, Martin, W. N. and Richards, D. S. [1991], A Multi-Population Genetic Algorithm for Solving the K-Partition Problem on Hyper-Cubes, *in* 'Proceedings of the Fourth International Conference on Genetic Algorithms', pp. 244–248.

Kennedy, J. and Eberhart, R. [1995], Particle Swarm Optimization, *in* 'IEEE International Conference on Neural Networks', pp. 1942–1948.

Lin, S.-C., Punch, W. and Goodman, E. [1994], Coarse-grain Parallel Genetic Algorithms: Categorization and New Approach, *in* 'Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing', pp. 28–37.

Mühlenbein, H., Schomisch, M. and Born, J. [1991], The Parallel Genetic Algorithm as Function Optimizer, *in* 'Proceedings of the Fourth International Conference on Genetic Algorithms', pp. 271–278.

Neri, F. and Giordana, A. [1995], A Parallel Genetic Algorithm for Concept Learning, *in* 'Proceedings of the Sixth International Conference on Genetic Algorithms', pp. 436–444.

Parmee, I. C. and Vekeria, H. D. [1997], Co-operative Evolutionary Strategies for Single Component Design, *in* 'Proceedings of the Seventh International Conference on Genetic Algorithms', pp. 529–536.

Pettey, C. C. and Leuze, M. R. [1989], A Theoretical Investigation of a Parallel Genetic Algorithm, *in* 'Proceedings of the Third International Conference on Genetic Algorithms', pp. 398–405.

Punch, W., Averill, R., Goodman, E., Lin, S.-C. and Ding, Y. [1995], 'Using Genetic Algorithms to Design Laminated Composite Structures', *IEEE Expert* **10**(1), 42–49.

Punch, W., Goodman, E., Pei, M., Chia-Shun, L., Hovland, P. and Enbody, R. [1993], Further Research on Feature Selection and Classification Using Genetic Algorithms, *in* 'Proceedings of the Fifth International Conference on Genetic Algorithms', pp. 557–564.

Schaffer, J. D., Caruna, R. A., Eshelman, L. J. and Das, R. [1989], A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, *in* 'Proceedings of the Third International Conference on Genetic Algorithms', pp. 51–60.

Schwefel, H. and Rudolph, G. [1995], Contemporary Evolution Strategies, *in* 'Third International Conference on Artificial Life', pp. 893–907.

Shonkwiler, R. [1993], Parallel Genetic Algorithms, *in* 'Proceedings of the Fifth International Conference on Genetic Algorithms', pp. 199–205.

Spears, W. M., DeJong, K. A., Bäck, T., Fogel, D. B. and de Garis, H. [1993], An Overview of Evolutionary Computation, *in* 'The Proceedings of the European Conference on Machine Learning', pp. 442–459.

Spiessons, P. and Manderick, B. [1991], A Massively Parallel Genetic Algorithm: Implementation and First Analysis, *in* 'Proceedings of the Fourth International Conference on Genetic Algorithms', pp. 279–286.

Syswerda, G. [1991], A Study of Reproduction in Generational and Steady State Genetic Algorithms, *in* G. J. Rawlins and M. Kaufmann, eds, 'Foundations of Genetic Algorithms'.

Tanese, R. [1987], Parallel genetic algorithms for a hypercube, *in* 'Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application', Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, pp. 177–183.

Tanese, R. [1989*a*], Distributed genetic algorithms, *in* 'Proceedings of the third international conference on Genetic algorithms', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 434–439.

Tanese, R. [1989*b*], Distributed genetic algorithms for function optimization, PhD thesis, University of Michigan. Co-Chairman-John H. Holland and Co-Chairman-Quentin F. Stout.

Tang, M. [2002], A Design Pattern for Web-based Parallel Genetic Algorithms, *in* 'TENCON '02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering', pp. 612–615.

Tsutsui, S. and Fujimoto, Y. [1993], Forking Genetic Algorithm with Blocking and Shrinking Modes (fGA), *in* 'Proceedings of the Fifth International Conference on Genetic Algorithms', pp. 206–213.

Wolpert, D. and Macready, W. [1990], 'No-free-lunch Theorems for Optimization', *IEEE Transactions on Evolutionary Computation* **1**(1), 67–82.