Exploring Applications of Extremal Optimization

by

Eric V. Drucker

(Under the direction of Walter D. Potter)

Abstract

Extremal Optimization (EO) is a relatively new single search-point optimization heuristic based on self-organized criticality. Unlike many traditional optimization heuristics, EO focuses on removing poor characteristics of a solution instead of preserving the good ones. This thesis will examine the physical and biological inspirations behind EO, and will explore the application of EO on four unique search problems in planning, diagnosis, path-finding, and scheduling. Some of the pros and cons of EO will be discussed, and it will be shown that, in many cases, EO can perform as well as or better than many standard search methods. Finally, this thesis will conclude with a survey of the state of the art of EO, mentioning several variations of the algorithm and the benefits of using such modifications.

Index words: Extremal optimization, Snake in the box, Forest planning problem, Mobile subscriber equipment, Multiple fault diagnosis

Exploring Applications of Extremal Optimization

by

Eric V. Drucker

B.S., The University of Georgia, 2007

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2009

Exploring Applications of Extremal Optimization

by

Eric V. Drucker

Approved:

Major Professor: Walter D. Potter

Committee: Khaled Rasheed
Pete Bettinger

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2009

Dedication

To my wife, Brooke, my parents, Bob and Joan, and my sister, Jenny. Thanks for all of your encouragement.

Acknowledgments

I would like to thank the AI faculty for always providing meaningful and interesting discussions on new and exciting AI applications. I would also like to thank my thesis committee, Dr. Rasheed and Dr. Bettinger, for helping me through to the very end of my degree. Finally, I would like to thank Dr. Potter for helping me achieve so much during my time UGA.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Search and optimization heuristics have played a large role in the field of Artificial Intelligence since its inception. Creators of many of these heuristics have turned to nature for inspiration, looking for examples where order is derived from chaos. Traditionally, two types of systems have been researched: physical and biological. This thesis will survey Extremal Optimization (EO), a recently developed optimization heuristic that has roots in both biological and physical systems.

Extremal Optimization [9] is a single search point (single solution) heuristic based on the Bak-Sneppen model of co-evolution [2]. The Bak-Sneppen model demonstrates self-organized criticality (SOC), a tendency for dynamical systems to organize themselves into optimal states. The Bak-Sneppen model addresses the evolution of entire ecosystems, or neighborhoods, via punctuated equilibrium instead of individual species gradually evolving over time. These so-called neighborhoods are made up of several species, each demonstrating complex relationships with neighboring species. Accordingly, every time the fitness of a species changes, the fitness of each one of its neighbors has the potential to change as well. EO individuals are based on these neighborhoods, and each component of the individual represents a different species within the neighborhood. Mutation is simulated by replacing the fitness of the least fit species in the neighborhood with a new, randomly-generated species. This change may affect the fitness of some or all of the selected species' neighbors, causing a chain reaction through the entire neighborhood [2]. After this drastic change, the neighborhood has the potential to be in a more fit state than it previously was. Hence, unlike most

optimization techniques, EO focuses on removing poor components of solutions instead of favoring the good ones.

This thesis will begin by defining several optimization heuristics to which EO will be compared, including Genetic Algorithms, Simulated Annealing, Particle Swarm Optimization, and Raindrop Optimization. Next, in Chapter 2, the underlying theories of EO will be discussed. Chapter 3 will then address the EO algorithm and mention a few common variations of the heuristic. Next, Chapters 4 and 5 will discuss new experiments performed comparing EO to the heuristics previously mentioned. Finally, in Chapter 6, this thesis will conclude with a survey of the state-of-the-art of EO, mentioning a few of the novel application areas in which it has been used.

1.1 Background

1.1.1 Genetic Algorithms

John Holland created genetic algorithms (GAs) based on the ideas in Darwin's theory of evolution. In simple terms, GAs recreate the ideas behind "survival of the fittest" to find optimal solutions to problems [18]. GAs consist of a population of individuals, or possible solutions to the problem being solved. Solutions are generally (and most easily) represented as binary strings, where each bit in the string corresponds to some property or action in the final solution.

Traditional GAs involve three operators to successfully run: selection, crossover, and mutation. Selection is performed to determine which individuals in the population should be used, and is based on the idea of natural selection, where the more fit individuals will be selected more frequently than the less fit individuals. Once individuals have been selected, crossover is performed. The idea of crossover is based on the crossover of chromosomes in genetic reproduction. Genetic material from each "parent" chosen in selection is combined to create new individuals having characteristics of each. Since better individuals are usually chosen in selection, crossover usually combines good characteristics to create individuals with

many more strong characteristics. Finally, mutation helps find better solutions by slightly changing information contained in each individual that might not otherwise be considered in a population. For example, if no individuals in a population had a specified trait, this trait would never exist in any children, unless this mutation of "genetic code" is performed.

1.1.2 Simulated Annealing

The SA algorithm [23] was developed to solve combinatorial optimization problems by mimicking the gradual cooling of extremely hot metals to harden alloys, and is based on the Metropolis algorithm [26]. The gradual cooling, based on probabilities found using a cooling schedule, is used to determine whether or not a randomly generated configuration should be accepted as the successor to the current configuration, regardless of its fitness value. When considering a more fit neighbor, it is always accepted as the successor. Using this probability function, as time progresses, the chances that a worse configuration will be accepted decreases, forcing the heuristic to mimic a greedy search. The possible successors of any given state are known as its neighbors, and typically are those reachable by modifying one component of the current configuration.

1.1.3 Particle Swarm Optimization

Particle Swarm Optimization [22, 21, 17] is a population-based optimization heuristic modeled on social behavior to demonstrate swarm intelligence. Swarm intelligence is based on the collective behavior resulting from systems made up of populations. Such behavior can be seen in nature in schools of fish, flocks of birds, and swarms of bees. Even in humans, social-psychological principles narrate social influences and learning mechanisms that occur, causing individuals to move towards and away from each other throughout "socio-cognitive space". When considering a flock of birds, each individual bird moves along its own independent path. However, the movement of the flock as a whole is determined by the direction and speed of the lead bird in conjunction with the speed and direction of each bird and its

actions based on its surrounding neighbors. PSO, like flocks of birds, moves towards a target based on the combined movement of many individuals within a population. The resulting movement of the swarm is caused by the movement of each individual particle. Each member of the population moves corresponding to the factors outlined below:

- ΔGB, the difference between the current location and the best global location seen.
- ΔPB, the difference between the current individual's best seen location and the current location.
- $\textit{Social influence} = C_1 \times Rnd \times \Delta PB$
- $\textit{Cognitive influence} = C_2 \times Rnd \times \Delta GB$
- $Velocity = \textit{inertia} \times (\textit{previous velocity}) + \textit{Social influence} + \textit{Cognitive influence}$

In each PSO iteration, each particle in the swarm is set to its previous location plus $Velocity$. The velocity is constructed from the social and cognitive influences, much like those in flocks of birds, requiring a balance of the two to be successful. If either is assigned too high a weight, the heuristic will fail to find a global optimum.

1.1.4 Raindrop Optimization

Raindrop Optimization [5, 36] is another recently developed single search point optimization heuristic. It was originally tailored for the Forest Planning problem described in Chapter 5, and makes use of both stochastic and deterministic methods to find good solutions. Although it is a single search point technique, RO is more invasive than the other heuristics discussed, in that it may alter a solution several times before calculating its fitness again.

In RO, raindrops and their subsequent rippling effect in bodies of water are simulated. To do so, a component in a solution is selected at random to be where the raindrop strikes, and the component is assigned a new random value. Adjacent components are then evaluated to determine whether or not their current values conflict with the newly assigned value. If so,

they are modified deterministically, and the same process is performed on all of its neighbors until no neighbors of affected components are in a conflicting state, thus simulating the end of the rippling effect. The RO algorithm is detailed below:

1. Generate an initial solution S, and set S_{best} equal to S.
2. Randomly select one of the components of S, S_c, and change its value randomly.
3. Assess whether any domain constraint violations have occurred.
4. If no domain constraint violations occur, go to step 7.
5. Otherwise, create a list of components of S that contribute to the constraint violations.
6. Repeat the following until the list is empty:
   (a) Select and remove from the list the component S'_c that is physically closest to S_c,
   (b) Deterministically change the value of S'_c so that the next best choice for it is used, and that the choice does not result in a constraint violation with components physically closer to S_c than S'_c.
   (c) If this value causes further constraint violations, add the affected components to the list.
7. If S is better than S_{best}, then set S_{best} equal to S.
8. Otherwise, if X iterations have passed with no improvement, reset S to S_{best}.
9. If N iterations have passed, then stop, otherwise return to step 2.

RO only uses two tunable parameters: the number of total iterations, and the number of iterations to perform before reverting to the best-seen configuration.

Chapter 2

Self-Organized Criticality & The Bak-Sneppen Model

This chapter will discuss the theories that led to the development of Extremal Optimization, particularly the self-organized criticality and Bak-Sneppen model of co-evolution. The Bak-Sneppen model outlines the co-evolution between several species in a particular neighborhood, or ecosystem [2]. The foundations of the Bak-Sneppen model lie in self-organized criticality (SOC), a tendency for systems occurring in nature to become organized in complex optimal states. SOC was originally discussed by Katz [19], however was popularized by the work done by Bak, Tang, and Wiesenfeld [4, 3]. SOC, as described by Bak [1], is "how a system that obeys simple, benign local rules can organize itself into a poised state that evolves in terms of flashing, intermittent bursts rather than following a smooth path". Dynamical systems generally demonstrate this property. Over time, such systems become organized in such a way that, when a small change is made to the state of a single property of that system, other parts of the system are disrupted and change states. This effect is known as an avalanche [1], and when key parts of the system change states, these avalanches can be large and can affect the majority of a system.

Sometimes, researchers describe the self-organized critical state as "the edge of chaos" [2]. This is because, in this state, the system is stable and inactive; however, a small change to the system will cause many other changes to occur, causing the system to become highly disorganized. This effect is best demonstrated in the sand pile experiment [1], which is performed as follows:

There is a small, flat table with nothing on its surface. Grains of sand are placed on top of the table one by one. Over time, a small pile of sand will form. Every so often, placing

a grain of sand on the top of the pile will disrupt a couple of grains and cause them to move down the pile, demonstrating a small avalanche. As the pile grows bigger, its overall slope will grow, which in turn causes the avalanches to become larger. The edges of the pile will eventually reach the edge of the table. Avalanches will still occur at both small and large levels, but they will not continue to grow in size, because the entire system can only hold a finite amount of sand. At this point, the system is in its self-organized critical state. Immediately following an avalanche and before another grain of sand is added, the system will be completely still; yet when a grain of sand is added, another avalanche is triggered, causing the system to be unstable again.

The concept of SOC applies to many different dynamical systems in the universe. It has been observed in plate tectonics, star quakes (the collapse of a part of a pulsar), solar flares, and most importantly, evolution [2, 1]. Bak and Sneppen developed their model of co-evolution as a way to describe how SOC may explain, among other things, the concept of punctuated equilibrium. Punctuated equilibrium is an evolutionary theory proposing that species evolve in rare, drastic events rather than gradually over time. The theory was introduced by Niles Eldredge and Stephen Jay Gould [16] as an alternative to the idea that species are progressively evolving over time, also known as phyletic gradualism, the theory commonly attributed to Charles Darwin's work. Eldridge and Gould claimed that in many cases, fossils show little to no "gradual" change over time, and that sudden, drastic changes are much more apparent. Similar to Darwin's theories, these mutations lead to fitter species, thus, SOC is a perfect phenomena to seek inspiration from in the development of an optimization heuristic.

Chapter 3

Extremal Optimization

Extremal Optimization is a heuristic based on the steps included in the Bak-Sneppen model of co-evolution. Boettcher and Percus used the rank based selection and mutation operators from the Bak-Sneppen model to set up their initial experiments on graph bi-partitioning, where it performed well enough to compete with well-established heuristics, including Simulated Annealing and Genetic Algorithms.

Graph bi-partitioning is an NP-hard problem, and was the first problem on which Boettcher and Percus chose to apply their newly created Extremal Optimization heuristic. Graph bi-partitioning is the problem of partitioning a set of N vertices of a graph into two equal subsets such that the number of "edges" (connections between pairs of points) that cross the partition of points is minimized. The number of such crossing edges is known as the cut-size; thus, the problem can be reduced to searching for the minimal cut-size of a graph [9].

Boettcher and Percus attempted to solve a graph bi-partitioning problem using the algorithm detailed below [9]:

1. The points in the graph are partitioned randomly into two equal subsets

2. Each point in each subset is assigned a fitness equal to $\frac{g_i}{(g_i+b_i)}$, where g_i and b_i are the number of good edges and bad edges, respectively. If a point has no edges, it is assigned a fitness of 1.

3. The point with the lowest fitness is selected, and a second point from the opposite subset is selected at random. These two points then swap subset membership.

4. Steps 2 and 3 are repeated until a stop criterion has been met.

The final result of the algorithm is defined in this case as the lowest cut size seen so far. In these initial experiments, EO performed well, finding the optimal partition several times. Boettcher and Percus note that EO performs well because it is able to evaluate many local optima, even late in the run, a characteristic that most general-purpose heuristics like SA and GAs lack due to their tendency to converge.

3.1 τ-FUNCTION

Unfortunately, step 3 of the original model can cause the heuristic to become stuck in local optima. To handle this problem, Boettcher and Percus introduced a power-law distribution approach for selecting the component to be modified [9, 7]. To select a component with this approach, an integer k is determined using the probability function

$$P(k) \propto k^{-\tau}; 1 \leq k \leq N$$

where N is the number of components in the configuration, and τ is a constant that determines how stochastic or deterministic the selection should be [7]. Figure 3.1 shows the selection probability for ranked components using a representation containing 100 components. Note that the lowest ranked components are heavily favored, but even the best-ranked component has a slight chance of being selected for mutation. Using τ-selection, the standard EO algorithm is detailed below:

1. Generate an initial value for each component of S, and set S_{best} equal to S,
2. Determine a fitness associated with each component of S,
3. Rank the components by fitness and select component k using τ-selection,
4. Assign the selected component a new random value to create S',
5. If the fitness of S' is higher than the previous best, S_{best}, replace S_{best} with S',

6. Set S equal to S',

7. Repeat steps 2 through 6 n times, or until stop criterion is met.

With higher values of τ, EO is more likely to get stuck in local optima for the same reasons it does without a selection function at all. For lower values of τ, EO sometimes replaces better components of a solution in order to explore a larger part of the search space. If τ is set to zero, EO produces similar results to random search, and if τ is set to ∞, then it resembles the original SOC-based EO. Because of this semi-stochastic nature, EO can jump in and out of near-optimal solutions at any point in time.

Admittedly, τ has no basis in nature; however, it has shown to have a significant impact on the chances that an optimal solution will be found by ensuring an appropriate amount of the search space is accessible, thus demonstrating ergodic behavior. Boettcher and Percus studied the effects of τ on the solutions found by EO, noting that it depends more on the number of mutations and solution components than it does on problem specific information [7]. This is explained by the fact that they found that optimal values of τ lay at the transition point where the heuristic moves from ergodic to non-ergodic behavior. These studies also found that an ideal value for τ can be found using the equation

$$\tau \approx 1 + \frac{\log(\frac{A}{\log N})}{\log N}$$

where N is the system size, and A is the number of cycles the EO algorithm is run. Thus, better optima can be found with smaller τ values when there are more possible solutions, and larger values of τ can be used for longer run-times. Using the equation to estimate values for τ, Boettcher and Percus found that using values that were higher or lower than the estimated value produced progressively worse solutions.

3.2 Comparisons with Simulated Annealing and Genetic Algorithms

To validate the use of EO in a field where researchers already make use of several other strong optimization heuristics, Boettcher and Percus compared the use of EO to Simulated

Annealing (SA) and Genetic Algorithms (GAs) on the graph bi-partitioning of four well-known graphs [6]. SA was chosen because it is also a single search point optimization heuristic, and has been used on problems in many domains for several years. EO differs from SA in that SA seeks out better components to a configuration whereas EO seeks to replace the worse components in a configuration. Another key difference between EO and SA is that EO only has at most one parameter to tune (τ), and SA allows for several due to its cooling-schedule technique. GAs were selected for comparison as well because they are among the most commonly used general optimization heuristics. EO differs from GAs in two significant ways. First, GAs are population based whereas EO only uses a single individual. Second, GAs focus on preserving strong genetic content by selecting the best individuals, and EO focuses on removing poor components from solutions in order to move the solution to a more optimal state.

During their experiments, Boettcher and Percus found that on many different graphs, EO was able to find optimal solutions faster than SA. Also, for similar problems, EO was able to find optimal solutions to similar problems with increasing representation sizes where SA produces worse results as the size increased. In some cases, a GA converged more quickly than EO, and in others, EO ran faster than a GA; however, both delivered equivalent, optimal results in a reasonable amount of time, thus proving EO's validity as a trustworthy optimization heuristic.

3.3 Generalized Extremal Optimization

One issue related to the use of EO is the method of calculating a fitness contribution for an individual component of a solution. Sometimes this is not even possible given certain criteria in some domains. To overcome this, de Sousa et al. introduced Generalized Extremal Optimization, or GEO [13]. GEO was designed for use on bit-strings. For each component bit in an individual, GEO flips the bit and calculates the new fitness. Each bit is then ranked by the resulting fitness, where higher fitness values are lower ranked. Though GEO causes

the heuristic to execute the objective function several more times (once for each bit of an individual), it allows EO to be used in many more domains, particularly those in which a tailored fitness contribution would not be appropriate.

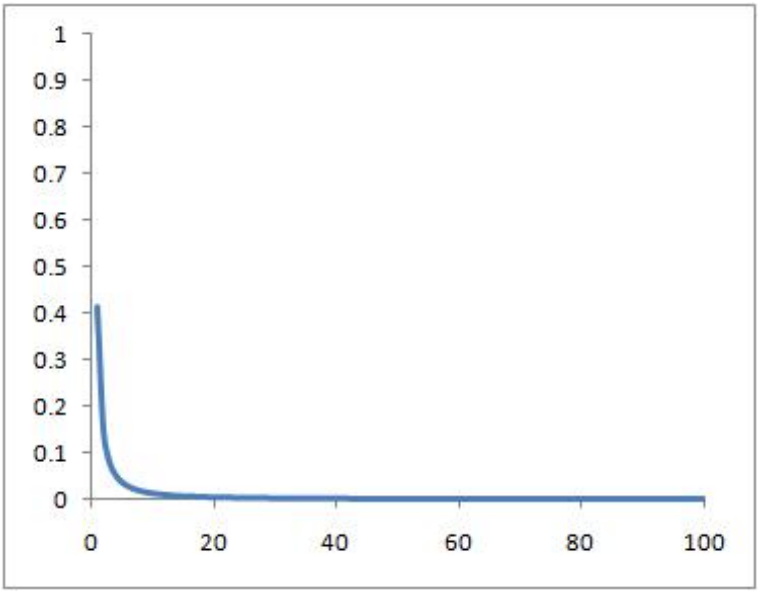

Figure 3.1: Selection probability distribution using a representation containing 100 components with τ set to 1.5

Chapter 4

Heuristic Evaluations Using the Mobile Subscriber Equipment Problem

In this chapter, we will compare EO, GA, PSO, and RO on the Mobile Subscriber Equipment (MSE) problem, a discrete network configuration problem. Some of the work in this chapter led to the publication of [25].

4.1 Mobile Subscriber Equipment

Mobile Subscriber Equipment (MSE) was once used in battlefields to provide a communications framework to wired and wireless subscribers in a military operation. An optimal list of equipment helped to provide the necessary communications without wasting equipment and manpower. Research using optimization heuristics on MSE demonstrated success [33], and although it is outdated, MSE is still valuable as a benchmark optimization problem.

Several heuristics were selected for comparison with Extremal Optimization applied to MSE, including Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Raindrop Optimization (RO). GA and PSO are both population-based heuristics, RO is a single search point heuristics. GA is the most frequently used, whereas PSO and RO are up-and-coming heuristics in their respective fields.

An MSE network consists of many components, including Node Centers (NCs), Large Extension Nodes (LENs), Small Extension Nodes (SEN1s, SEN2s), System Control Centers (SCCs), NATO Analog Interfaces (NAIs), and Remote Access Units (RAUs). NCs acted as the backbone, allowing the various other components to connect to the network. The different extension nodes (LENs, SEN1s, and SEN2s) connected different numbers of wired subscribers to the network, and RAUs connected wireless subscribers. Mission requirements involving

MSE would only designate the number of wired subscribers (MSRTs) and wireless subscribers (DNVTs), thus the configuration of an MSE network involves determining how many of each component to use for a given mission. The resulting representation for optimization heuristics is an integer sequence, one integer value for each type of component.

The objective function will determine how well a given set of components will work for the desired number of subscribers. If the set of components will absolutely not work, this function will result in a value of zero. Otherwise, the function will result in a positive number, where a higher number is a better fit for the set of components to the desired number of wireless and wired subscribers in the network.

The following variables are used throughout the objective function. N represents the total number of antennas needed to support the network; TA represents the number of antennas available in the network, which is twelve times the number of Node Centers in the list of components; and AFB is the total number of antennas available once the backbone of the network is set up.

$$\begin{aligned} N &= TA - AFB + (others) \\ TA &= Components_{NC} \times 12 \\ AFB &= (32 \times Y) - X^2 - (13 \times X) \\ X &= ((Components_{NC} - 1)\ mod\ 4) + 1 \\ Y &= \frac{(Components_{NC} - 1)}{4} \\ Z &= Y + 1 \end{aligned}$$

Once these variables have been determined, the following seven terms can be calculated and used to determine the overall value for the objective function. The cardinality term, defined below, is used to give higher values to configurations that use fewer components. $Components_i$ represents the number of components of type i (NC, LEN, SEN1, SEN2, SCC, RAU, or NAI) in the input configuration. $Corps$ represents the number of components

available to be used, and P is a predetermined constant (equal to 1/7) to give each component an equal weight in the function.

$$Cardinality\ Term\ =\ \frac{50}{\sum_{i=0}^{\#Components} \frac{Components_i}{Corps_i} \times P}$$

The SEN1 to SEN2 relationship is used to let the function favor configurations that have a ratio of 3 SEN2s to every SEN1. Thus, its value is equal to the lesser of either the number of SEN1s divided by three times the number of SEN2s, or three times the number of SEN2s divided by the number of SEN1s. It has been previously determined that this is an optimal ratio based on support and resources required.

The UHF connectivity is used to favor network configurations that have fewer excess antennas. If there aren't enough antennas to support the network, the overall objective function will later prove to be invalid, but if there are many extra antennas, there is a waste of resources, which should be avoided. The UHF connectivity term is determined by dividing the number of needed antennas by the number of available ones if the total number available from the backbone is greater than 12 more than the number needed, otherwise it is equal to one. This will lower the total objective value with a larger number of antennas available from the backbone.

The minimum constraint violation is used to ensure that there is at least one node center and at least one system control center in the configuration because at least one of each is required in a valid network. It is assigned a value of zero if either of these components is missing, otherwise it is assigned a value of one. The maximum constraint violation is calculated to determine whether or not the configuration has a valid number of each type of component. If more of any type of component exists within the configuration than there are available for use, the term is set to equal zero and the configuration is determined to be invalid.

Lastly, the MSRT support term and the DNVT support term represent whether or not the required number of MSRTs and DNVTs can be supported by the network. If either has a required value that is larger than the number supported by the network, the configuration is determined to be invalid by assigning the appropriate support term a value of zero. Otherwise, the support terms are calculated by dividing the required number of subscribers by the number supported by the configuration. This favors configurations that support a number of subscribers that is as close to the number required as possible, in order to avoid wasting resources.

The overall objective value is then calculated by multiplying each of the seven terms together. If any term resulted in a value of zero, the objective value is equal to zero and the configuration is invalid based on the requirements of the network.

4.2 Directional Extremal Optimization

In initial experiments on MSE we used weighted component-specific evaluation functions, and when a component was selected for mutation, it would be replaced by a new random value. This process was very disruptive to good solutions, and would often result in invalid configurations. GEO has proven itself extremely useful in situations where a unique fitness evaluation cannot be determined; however, it requires bit-strings to work correctly. Thus, the concept of GEO was applied using integers instead of bits, using additional weights to create a directional scheme. In this scheme, to determine the amount in which a particular value is "holding back" the overall fitness, each component is evaluated twice, once with its value incremented, and once with it decremented. The component is then assigned a direction in which it should be changed based on which value provided a better fitness. If both the incremented and decremented values produce the same fitness as the original value, or if they are both lower than the original fitness, then no direction is assigned. Each component is then ranked by the new fitness in its selected direction, and if no direction was assigned, the component's original fitness was used. Selection is then performed normally, either using the

worst ranked or using τ-selection. Finally the selected component is assigned a random value between the original value and the bound in the selected direction, or a random value between the upper and lower bound for those components that were not assigned a direction. Because EO is designed to weed out the poor components of a solution, the direction scheme works particularly well in discrete-integer problems by adding a positive bias to newly assigned values for solution components.

4.3 Setup

The GA was set up using tournament selection, point crossover, and an age proportional mutation operator. Tournament selection and point crossover are both standard GA operators. In tournament selection, two or more individuals in the population are selected at random, and the one with the highest fitness then goes on to crossover. Once two individuals are selected in this manner, crossover is performed. Point crossover is carried out by selecting a point within the representation at random and then combining opposite halves to create two new individuals. Finally, mutation is carried out using the age proportional operator devised for this application. For each component selected for mutation, a new random value is generated and then divided by the age of the particular individual. This operator allows more disruptive mutation early on and prevents mutating away from better solutions as time passes.

For the GA, trials were run using populations of 300, 400, and 500, and using mutation rates of 5%, 10%, and 15%. On average, the GA ran for 34 generations using nearly 14,000 fitness evaluations. Increases in the mutation rate always demonstrated increases in fitness regardless of population size, so further experiments were performed only using a mutation rate of 15%.

The PSO algorithm was evaluated using 8 parameter settings. Swarm sizes of 100 and 300 were used with $C_1 = 1.5$, $C_2 = 4.0$, and $0.4 \leq inertia \leq 1.0$, where *inertia* was incremented in steps of 0.1. Also, V_{min} was set to -6 and V_{max} was set to 6.

To evaluate MSE using RO, we considered it a constraint violation when the required number of either wired or wireless subscribers was not met, ie. when the fitness was equal to zero. Values for each component were incremented and decremented appropriately for each iteration as long as the configuration remained valid. We performed our experiments using iterations in a range from 5,000 to 20,000, and the number of reversions set to either 4 or 5.

The EO algorithm was set up to evaluate each component of the MSE component list. Each component was assigned a penalty according to each goal of the objective function. Since some components could be affected by more goals than others, the penalties were normalized before the components were ranked. Once ranked, a power-law distribution with τ equal to 4 was used to select the component to be modified, and the selected component was assigned a number between 0 and the maximum possible value. An initial trial using this as the only evaluation criteria resulted in a reliability of less than 10% on 1,000 runs of 10,000 EO iterations each. The results of the original experiment demonstrated a need for a less drastic change in the numbers of components, because the heuristic would sometimes become stuck. To improve the component evaluation function, Directional Extremal Optimization was implemented. Using directions, an innovative adaptation of the EO approach, reliability was drastically improved to 92% on 1,000 runs.

The EO algorithm was then run 1,000 times for each of 3,000 parameter settings. The parameters were τ (0 to 10 in increments of 0.01) and number of iterations (1,000, 5,000, 10,000). The best value of τ was found to be in the range of 4.94 to 5.03. Using τ equal to 5, the EO algorithm was run on a range of set iterations (100 to 20,000 in increments of 100), and each set number of iterations was run 1,000 times. The maximum reliability reached was 100% for 19,700 iterations. On average, EO found the best solution after 4,000 iterations, and 57,300 fitness evaluations. Limiting the number of iterations to 10,000, cutting run-time in half, produced a reliability of 92.2% with the best solution found after an average of 3,200 iterations and 45,700 fitness evaluations.

4.4 Results & Observations

The results of our experiments on the MSE problem can be found in Table 4.1. GAs were run using a mutation rate of 15% with population sizes of 300, 400, and 500 (GA300, GA400, and GA500 respectively), and the highest reliability, 99.6%, was found using GA500. The GA500 required an average of 12,000 fitness evaluations to complete, whereas the GA400 and GA300 required 9200 and 6200 respectively. Although the reliability decreased slightly with smaller population sizes, the required time and number of required fitness evaluations dropped off significantly, demonstrating a more efficient performance with smaller population sizes.

PSO was run with swarm sizes of 300 and 100 (PSO300 and PSO100 respectively), using the tuned parameter settings. Similar to the results found using GAs, a larger swarm size resulted in a higher reliability. Again, the average number of fitness evaluations dropped off significantly with the smaller swarm size while maintaining a relatively high reliability.

RO was run with varying numbers of iterations (5,000 to 20,000), and performed with the highest reliability when using 20,000 iterations. Although RO only achieved a reliability of 72%, the lowest of the four heuristics, this is still an acceptable value for MSE. Also, RO was one of the more efficient heuristics, requiring only 14,000 fitness evaluations on average to find an optimal solution.

Finally, EO was used with a set number of iterations equal to 10,000 and 19,700 (EO10,000 and EO19,700). EO demonstrated similar characteristics regarding decreasing numbers of fitness evaluations and reliabilities. However, due to using the full fitness function to evaluate each component for each iteration, EO required a significantly larger number of fitness evaluations compared to PSO and GA. With further research on a less complex, more domain specific component fitness evaluation, the number of fitness evaluations could be reduced such that EO could compete with GA and PSO in terms of run-time. Let it be noted, though, that EO was the only heuristic to achieve 100% reliability on the MSE problem.

Overall, both EO and PSO performed well when compared to GA; however, neither heuristic could reach the efficiency of GA given that the ratio of reliability to the number of required fitness evaluations was much lower for both. In all cases, the ideal parameters are those that have the highest efficiency, not the best reliability, and the heuristic can be run several times to ensure an optimal value is found.

Table 4.1: Results seen in comparing GA, EO, and PSO on the MSE problem

Heuristic	% Reliability	Fitness Evaluations	Ratio
GA500	99.6	12,000	8.3000×10^{-3}
GA400	99.1	9,200	1.0772×10^{-2}
GA300	98.1	6,200	1.5823×10^{-2}
EO19,700	100	57,300	1.7450×10^{-3}
EO10,000	92.2	45,700	2.0180×10^{-3}
PSO300	99.85	31,500	3.1700×10^{-3}
PSO100	98.63	11,800	8.3580×10^{-3}
RO	72	14,000	5.1428×10^{-3}

To get a deeper understanding of the performance of EO on MSE, a few additional tests were performed. So far, we have only considered the number of fitness evaluations before termination when discussing the number of fitness evaluations performed. Figure 4.1 shows the average number of fitness evaluations before the optimal solution is found. Even though EO required an average of 45,700 fitness evaluations when running for 10,000 iterations, Figure 4.1 shows that EO found the optimal solution after an average of 3300 fitness evaluations. Consequently, one could argue that a better stop criterion could be created, or some sort of reversion mechanism (as in RO) could be introduced to keep EO from exploring dead ends. Figure 4.2 shows the reliability of EO as the number of iterations increases. Naturally, as the iterations increase, so does the reliability. However, for EO to achieve higher reliabilities, it requires exponentially more iterations to run. Thus, instead of using extremely high numbers of iterations, it may prove effective to perform multiple runs of EO. These values, combined with the values shown in Figure 4.1 may help find an optimal number of iterations for each run before starting over.

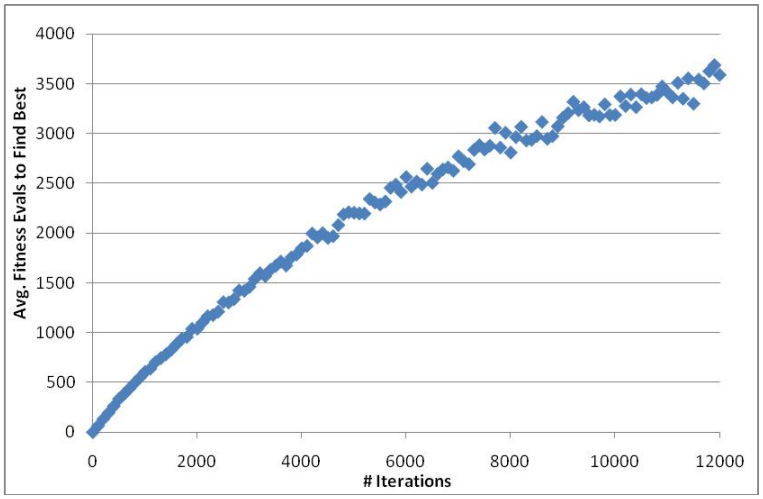

Figure 4.1: Average fitness evaluations to find S_{best}

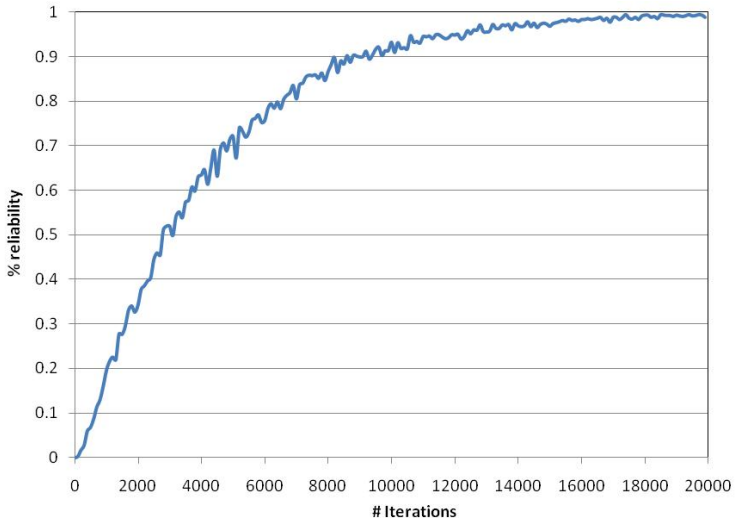

Figure 4.2: Reliability on MSE for $\tau = 5.0$, 0 to 20,000 Iterations

Finally, a quick study was done on the effect of different τ values on the reliability of EO. The MSE setup was used again, using 10,000 EO iterations, and running our algorithm repeatedly for $\tau = 0$ to $\tau = 10$, incrementing τ by 0.01 each time. Unsurprisingly, the results displayed in Figure 4.3 show that lower values of τ kept our algorithm from finding optimal solutions due to selecting random components too frequently. Similarly, values of τ that were too high also prevented our algorithm from finding good solutions by selecting the worst components too frequently. The resulting curve allowed us to find an optimal value of τ when using 10,000 EO iterations on this problem. These results confirm the conclusions drawn about τ in Chapter 3. As the number of iterations increases, the curve in Figure 4.3 reaches its peak more quickly and tapers less on the right side of the peak. For exceedingly high numbers of iterations, this curve resembles more of a plateau, allowing many values of τ to produce optimal results, given that a larger-than-necessary portion of the search space is being considered.

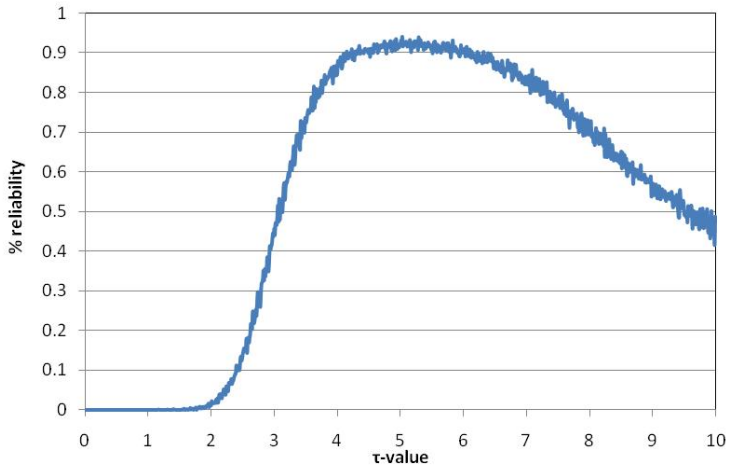

Figure 4.3: Reliability on MSE for τ from 0 to 10 with 10,000 Iterations

Chapter 5

Experiments in Diagnosis, Planning, and Pathfinding

After having success in comparing GAs, PSO, RO, and EO on the MSE problem, we extended our research to compare these heuristics in different application domains. To continue with our comparisons, we selected problems in diagnosis, planning, and pathfinding domains. For the diagnosis problem, we selected Multiple Fault Diagnosis. Next, for the planning problem, we selected the Forest Planning problem. Finally, for pathfinding, we chose the snake-in-the-box problem.

5.1 Diagnosis

Commonly used in fields such as medicine and physics, Multiple Fault Diagnosis (MFD) is the process of identifying a set of one or more causes to a given set of symptoms. As the number of possible causes increases, the number of possible diagnoses increases exponentially, demonstrating a need for an efficient method to make efficient, accurate diagnoses. Research has been performed using optimization heuristics successfully in such a manner [24, 32], making MFD a useful benchmark problem in comparing optimization heuristics.

For our experiments, we implemented Peng and Reggia's probabilistic causal model (PCM) [29, 30]. In the PCM, multiple fault diagnosis problems are identified by the following set:

$$\{D, M, C, M^+\}$$

where D is a set of disorders, M is a set of manifestations (symptoms), C is a set of (d, m) pairs, where disease d has some probability of causing symptom m, and M^+ is the subset of symptoms in M that a patient exhibits. Diagnosis DI is a subset of D, which identifies

the diseases potentially causing the symptoms in M^+. DI is easily represented by bit strings in optimization heuristics, setting a value of 1 if a disease is present in DI, and a value of 0 if it is not. To evaluate the fitness of DI, we made use of a modified version of the relative likelihood function as described in [29, 30]. The relative likelihood function utilizes a tendency matrix, in which each cell represents the likelihood that the corresponding disease causes a particular system. The information within this matrix is usually based on historical trends, and/or detailed analysis by domain experts. The full fitness value is the product of multiplying three values, $L1$, $L2$, and $L3$, detailed below:

$$
\begin{aligned}
L1 &= \prod_{m_i \,\epsilon\, M^+} \left(1 - \prod_{d_j \,\epsilon\, DI} (1 - c_{ij})\right) \\
L2 &= \prod_{d_j \,\epsilon\, DI} \prod_{m_i \,\epsilon\, \text{effects}(d_j) - M^+} (1 - c_{ij}) \\
L3 &= \prod_{d_j \,\epsilon\, DI} \frac{p_j}{(1 - p_j)}
\end{aligned}
$$

$L1$ represents the likelihood that the diagnosis covers the symptoms in M^+. $L2$ represents the likelihood that the diagnosis does not cover more symptoms than those included in M^+. Finally, $L3$ represents the likelihood that a common disease in the diagnosis contributes significantly to the likelihood of the overall diagnosis. For our experiments, we created an MFD problem consisting of 25 causes in the causal set, and 10 symptoms in the symptom set.

5.2 Planning

The goal of the FP problem is to optimize timber harvests for a single harvest schedule given several constraints. Each forest stand in a valid solution may be harvested only once out of the n harvest periods in the schedule, and adjacent stands may not be harvested in the same time period. Within the forestry domain, these restrictions help to prevent erosion among other things by preventing over-harvesting a forest. For the experiments discussed in this thesis, a problem in [5] was used that contained 73 forest stands. In this problem, harvest schedules were evaluated by the following formula:

$$Minimize \sum_{i=0}^{n} (H_i - T)^2$$

$$\begin{aligned} n &= \textit{number of harvest periods} \\ i &= \textit{harvest period} \\ H_i &= \textit{total harvest in period } i \\ T &= \textit{target harvest volume for entire forest per time period} \end{aligned}$$

All harvest volumes are measured in MBF, or thousand board feet. In this case, we used $n = 3$ periods, and a target period harvest volume $T = 34,467$MBF. To represent a harvest schedule, an integer sequence with a length equal to the number of stands is used, and each value in the sequence corresponds to the time period, 0 to n, that particular stand should be harvested.

5.3 Pathfinding

The Snake-In-The-Box (SIB) problem is the computationally difficult problem of finding the longest induced path through a d-dimensional hypercube. Kautz originally introduced the SIB problem as a method of finding unit-distance error-checking codes [20]. Since then, several researchers have found the longest induced paths in hypercubes up to dimension seven. Long paths have been found in higher dimensions; however, it is unknown as to whether these are the longest existing paths. Thus, the problem is still open for further research.

In introducing the SIB problem, Kautz discusses binary sequences characterized by two distinct traits: error-checking, and unit-distance. Successive elements in these sequences differ by only one bit. Long sequences of this type provide good uses in error-detection in converting analog into digital data, and in circuitry design. A way of modeling these special sequences is with n-dimensional unit cubes. Each vertex in the code is assigned a binary number such

that all of its neighboring vertices differ by only one bit. These sequences would then be paths through the n-dimensional hypercube. Additional constraints are then added, such as not allowing a path to visit the same vertex twice, and not allowing the path to go through any of the vertices adjacent to any previous vertex in the sequence. It has proven to be increasingly difficult to find long paths in higher dimensions, but longer sequences of this type would be more beneficial in several applications. The properties that this sequence demonstrates, particularly that it creates a tube-like path throughout the cube, are the reason the problem is called the snake-in-the-box problem [20].

Initial solutions to the problem were found using only mathematical proof, but this only worked in lower dimensions. Kautz provides solutions for hypercubes in dimensions up to n=6, but further solutions were not found until later. In higher dimensions, traditional formulas and exhaustive search do not work because of the exponentially increasing search space. This requires us to use alternate search methods to find good solutions. In dimensions 7 and above, the longest solutions have been found using evolutionary heuristics, mainly genetic algorithms [35, 34, 10]. For dimensions 8 and above, the longest solutions discovered are only lower bounds and have not been proven to be the optimum solution.

For our experiments, we chose to search for snakes in dimension 8. Although there are several possible representations for the SIB problem, we chose to use the bit string representation, as introduced by Diaz-Gomez [14, 15], and the transition sequence representation.

The bit string representation contains one bit for each node within a hypercube. For each bit, it is assigned a value of 1 if it is included in the snake or a value of 0 if it is not. To evaluate the fitness of a bit string, the length of the snake is determined by connecting neighboring nodes that are included, beginning with node 0. To ensure that the resulting snake is valid, Diaz-Gomez introduced a "vector of neighbors", created by multiplying the bit string representation with the adjacency matrix of the hypercube being used. The resulting vector includes the number of each nodes neighbors that are included in the snake, thus if any node has a value greater than 2, the snake is invalid.

The transition sequence representation is an integer sequence, where each value represents the bit that should be flipped to get to the next node in the sequence. In other words, given an adjacency matrix for a particular hypercube, the next node in the path is found in cell (n, t) in the matrix, where n is the current node number, and t is the transition value in the sequence. For both of these representations, snake length is the most frequently used fitness function; however, several fitness evaluation methods exist, specifically the "tightness" of a snake [35].

5.4 Problem Setup

This section will detail the setup for EO on each problem mentioned. For the other heuristics, the standard algorithms were used, and any modifications to the representations or evaluation functions will be mentioned briefly in the results section.

5.4.1 EO-MFD

For the MFD problem, Generalized Extremal Optimization (GEO) was used. Two experiments were performed: one to find the parameter settings leading to the highest efficiency measure (for comparison to other EO runs) and one to find the parameter settings leading to the highest reliability (recall, reliability is the ratio of optimal diagnoses found to the total number of diagnostic trials 1023). To find the highest efficiency measure, GEO was used with $0 \leq \tau \leq 6$ in increments of 0.01, and the number of iterations from 10 to 40 in increments of 1. To find the parameters leading to the highest reliability, GEO was used with $0 \leq \tau \leq 3$ (in increments of 0.01), and the range of total GEO iterations went from 100 to 500 (in increments of 5). Representing a diagnosis followed the other setup schemes presented, namely, a bit string representation. The modified relative likelihood fitness function served as the basis for evaluating individual solutions as well.

Our GEO algorithm had the highest efficiency measure using $\tau = 4.66$ and after 16 iterations, producing a reliability of 77.1%. We were able to achieve 100% reliability with

$\tau = 1.39$ and after 490 iterations. For comparison, the highest efficiency measure found was 0.048 however the run with the highest reliability had an efficiency measure of 0.002.

5.4.2 EO-FP

During our Forest Planning experiments, the issue of whether or not to include the "no-harvest" time period came into question due to the nature of EO. Consequently, we tried two similar approaches on the FP problem. The experiments were identical, but the second experiment allowed for stands to be unscheduled (i.e., not harvested) and the first did not. Allowing for stands to be left without being harvested allows the EO algorithm to move around in the search space, since there are more valid solutions. EO was used with an integer representation, using the values 1 to 3 for the first experiment (each corresponding to which harvest time period to choose), and 0 to 3 for the second experiment where a zero for any stand meant it was chosen to not be harvested. In both experiments, τ was tested on the range of $1 \leq \tau \leq 3$ (in increments of 0.05), and the number of iterations was set at 50,000. No tests were done on the efficiency measure because the optimal solutions are not known; only target best known solutions are available [5]. Instead, the tests were to find the best value of τ to use in further research on the problem. For the first experiment, no solutions below 20M were found, and there was no obvious improvement for any values of τ. For the second experiment, the best configuration found had a solution of 10,597,074 using $\tau = 1.5$.

5.4.3 EO-SIB

As with the EO-FP setup, the nature of EO led us to try two different representations on the SIB problem in our efforts to find a better known solution. The first was a bit string representation. EO was used with τ in the range of $1 \leq \tau \leq 3$ (in increments of 0.05). Again, because there are no known optimal solutions, the number of iterations was fixed at 10,000. The second representation used was a transition sequence representation.

Since an integer sequence was used, EO was modified to test more possible configurations. A transition sequence of length 100 was used, and for each transition, the fitness function was run 7 times, once for each different value the transition could be (not including its current value). Thus for each iteration, the fitness function was run 700 times. The new fitness values were ranked and their corresponding transitions tracked, and τ-selection was used on all 700 possible changes. This scheme was then modified to only keep track of the transitions that were part of the current snake, since no changes after that would affect the fitness of the new configuration.

With both representations, snake length was the primary indicator of snake strength. But, as with other search schemes where we found disappointing SIB results, we also considered tightness (nodes in the snake that share more neighbors with others are more fit than those that do not), and the number of free nodes (nodes that are not part of the snake path or adjacent to the snake path). The longest snake found using the transition sequence was of length 70 using $\tau = 1.4$, and the only parameter used in the fitness function was the length of the snake. The longest snake found using bit-string representation was of length 74, using $\tau = 1.6$.

5.5 Results & Observations

After the initial studies on MSE, all four heuristics were compared on the MFD, FP and SIB problems. The results of these comparisons can be found in Table 5.1. Although it may be argued that any of these heuristics could have performed well given enough time or a large enough population, these results here are representative of when can be achieved using normally tuned parameters for each heuristic.

On the MFD problem, a GA, DPSO, and EO performed very well, and all demonstrated the expected increase in reliability with an increased population size or increase number of iterations. RO did not have a good performance at all, achieving a reliability of only 12%, raising questions as to what aspects of the MFD problem held it back. One thing to note

relating to RO is that it only works with valid solutions, thus its diagnosis must completely explain some of the symptoms in question, whereas the other search techniques allowed for partial covers of the symptom set. Thus, the other search techniques were able to move more freely through the search space, possibly accessing areas that were difficult for RO to access with the given constraints.

Table 5.1: Results seen in comparing EO, GA, PSO, and RO on Multiple Fault Diagnosis, Forest Planning, and the Snake in the Box problem

Heuristic	MFD	FP	SIB
Extremal Optimization	100%	10M	74
Genetic Algorithm	87%	6,505,676	95
Particle Swarm Optimization	98%	35M	86
Raindrop Optimization	12%	5,500,391	65

For the Forest Planning problem, using RO we were able to reproduce the optimal results seen in [5]. GAs were the only other technique to achieve a decent schedule for FP, though a modified fitness function was introduced to add a penalty to solutions with constraint violations. This modified fitness function allowed the GA to preserve potentially strong genetic material. Both DPSO and EO struggled in finding optimal solutions for FP, demonstrating an opportunity for future research to be performed.

Finally, all heuristics performed well on the SIB problem. Using the bit-string representation, GAs found the best snake of length 95 using only 100 individuals and running for 250 generations. The second best snake, which had a length of 86, was found using DPSO. DPSO found its best using a transition sequence representation. Also, DPSO used seeding, setting the initial swarm equal to long snakes in smaller dimensions rather than a purely random initialization. EO and RO both found their longest snakes (74 and 65 respectively) using the bit string representation. Although these snakes were significantly shorter than the one found by GA, the two heuristics performed reasonably well, given that they are both single element search methods, and that a length of 65 can be considered average [31].

Overall, EO performed well on the problems with a smaller search space (MFD and MSE), yet struggled on the harder problems. Our speculation is that this is caused by the

use of a general evaluation function for each component in our representations rather than using problem-specific information. For the FP problem, we could have potentially held back EO by using the "no-harvest" time period. Even though this allowed EO to move more freely through the search space by providing more valid configurations, it drastically increased the size of the search space, making the problem even harder to solve. For our SIB experiments, we may have held back our solutions by unfairly favoring those nodes near the beginning and end of the snake. While this helped to find medium-length snakes, no information was considered regarding factors other than length, so sub-optimal sequences towards the middle of the snake would not be modified, holding EO in sub-optimal areas of the search space. This "holding back" could be prevented with an increased τ value; however this could resemble more of a random search than a good search heuristic. In both cases, a problem-specific evaluation for each component of a solution would have avoided these negative characteristics, potentially yielding better results.

Chapter 6

EO Hybridization & State of the Art

So far, this thesis has only discussed the standard EO heuristic and a few notable modifications. This chapter will discuss the state of the art of Extremal Optimization, including new adaptations and instances in which it has been hybridized with other heuristics. EO has performed very well in several domains, and using these state of the art adaptations, it has the potential to perform even better.

6.1 Population-Based Extremal Optimization

To improve on the performance of EO, Chen, Lu, and Yang introduced population-based EO (PEO) [12]. Initially, PEO is described as performing the EO algorithm on a set (population) of solutions, helping to explore a larger portion of the search space in parallel. To further enhance performance, Chen et. al. introduced a Lévy mutation operator to EO. The Lévy mutation is an adaptable operator, combining the benefits of Cauchy mutation, which performs well on search points far from optimal solutions, and Gaussian mutation, which performs well when candidate solutions are near-optimal. Once the worst component of a solution was selected, this mutation was applied, and if the resulting value was valid, it was accepted. The combination of the mutation operator and the use of a population increased the speed at which the algorithm converged, and demonstrated a higher accuracy than other heuristics on several benchmark problems.

6.2 Continuous Extremal Optimization

Continuous Extremal Optimization (CEO) was introduced by Zhou, Bai, Cheng, and Wang to apply the benefits of EO on continuous optimization problems [37]. CEO consists of two parts; classical EO as a global search, and a local hill-climbing technique. CEO is performed in the same manner as EO; however, before each iteration it moves the solution to a local optimum using a hill-climbing technique before evaluating the components of the solution. To demonstrate a simple model using CEO, Zhou et al. implemented it on a Lennard-Jones optimization problem. In their experiment, they compared results using CEO to those found using several domain-specific methods as well as other general optimization heuristics. Zhou et al. found that although CEO didn't have great success on the Lennard-Jones problem when compared to the domain-specific methods, it outperformed all other general optimization heuristics.

6.3 Jaded Extremal Optimization

To improve the performance of EO, Middleton introduced Jaded Extremal Optimization (JEO) [27]. JEO incorporates an aging parameter into τ-EO, adjusting the fitness of each component proportional to the number of times it was previously been selected. The following equation is used to determine the value to add to the fitness evaluation of each component i:

$$\lambda_i = \lambda_i^{\Gamma} \equiv \lambda_i^0 + \Gamma k_i$$

Note that Γ is a tunable parameter in JEO, thus when $\Gamma = 0$, it is the equivalent of τ-EO. Middleton also notes that tuning Γ can be done fairly quickly, and it reduces the effort required to tune τ. In comparisons with τ-EO, JEO converged significantly faster, providing equivalent optimal results.

6.4 Particle Swarm Optimization Hybridization

PSO has been a useful addition to the wide range of available optimization heuristics, though it demonstrates some problems with early convergence. Chen et. al. hybridized PSO with EO to take advantage of the strong global search characteristics of PSO while using the effects of EO to keep the heuristic from becoming stuck in local optima [11]. The PSO-EO hybridization used a standard PSO model, but incorporated a population-based EO iteration for one out of every N PSO iterations, where N was a tunable parameter. Chen et. al. used their heuristic on six benchmark problems, each of which contain solution landscapes with many peaks and valleys, providing many points for traditional optimization heuristics to become stuck. Their tests using these benchmark functions resulted in a performance better than that of PSO, PEO, or GA alone, both in terms of convergence speed and success rate.

6.5 Dynamic Optimization

EO has also demonstrated success when applied to dynamic function optimization problems [28]. Moser and Hendtlass recently introduced multi-phase multi-individual Extremal Optimization (MMEO) and compared its use to standard optimization heuristics on the Moving Peaks (MP) problem. MP is a dynamic optimization problem that has been frequently used as a benchmark for testing optimization heuristics. Its fitness function changes over time, and as a result, the fitness landscape consists of several peaks, all of which change in height, width, and location over time. In initial experiments, Moser and Hendtlass found that the standard EO algorithm would not perform well on the MP problem. This was explained by the fact that the most successful algorithms applied to MP keep a memory of both local and global optima. As a result, Moser and Hendtlass devised MMEO to incorporate these missing features. The MMEO algorithm is performed as follows:

1. Use stepwise sampling to determine an initial value for each solution S_i in the population. Evaluate each solution, and rank them according to their fitness.

2. Use τ-selection to select solution S_i for mutation
3. Use a hill-climber to locally optimize S_i, halting optimization if S_i becomes too close to another solution
4. Store S_i if it is not a duplicate of any other solution
5. Reoptimize using hill climber if the fitness of any population member changed.
6. Use precise steps in the reoptimization of S_{best}, halting at convergence.

Moser and Hendtlass noted that the power-law selection produced poorer results than choosing the best individual each time, and continued their work using τ set to infinity. Using this basic setup, MMEO produced strong results on the MP problem. Moser and Hendtlass discuss many variations to the heuristic, and conclude that MMEO found great solutions for the MP problem; however it used a lot of problem specific information and didn't perform as well when using a more simplistic approach.

Chapter 7

Conclusions

Extremal optimization is an exciting addition to the optimization field. Its approach of "weeding out" the bad components instead of the traditional approach of favoring the good components has proven to be a positive search method. Because of its uncommon ability to transition in and out of near-optimal solutions at any point in time, EO is a great heuristic to exploit when considering a hybrid approach with any search technique, particularly those that frequently demonstrate signs of early convergence. Also, EO has performed well in a variety of domains, particularly fault diagnosis, network configuration, and path finding, showing its independence from problem specific information.

Due to the large number of variations of EO, there is a wide variety of potential research that could be performed in the future. Particularly, further work can be done on optimal parameter tuning for τ, and potentially looking into alternative probability functions in which to use τ. Also, more work can be done with the ideas behind Jaded EO, incorporating penalties for components of a given solution that are derived outside of the problem domain.

As far as our particular problem areas, more work can be done on finding good methods of evaluating the individual components of a solution for both the Forest Planning problem and the Snake in the Box problem. Although generic fitness evaluations helped find decent results, incorporating more problem specific information or using a different variation of EO could potentially find even better solutions. Even in the Mobile Subscriber Equipment problem, more work could potentially be done in evaluating the individual components in order to reduce the number of times the overall solution evaluation is run. Although EO

didn't produce outstanding results in all domains, it is definitely a heuristic worth using when attempting to solve a discrete optimization problem.

Bibliography

[1] Bak, P. (1996). *How Nature Works*. New York: Springer-Verlag.

[2] Bak, P., & Sneppen, K. (1993). Punctuated Equilibrium and Criticality in a Simple Model of Evolution. *Physical Review Letters*, 4083-4086.

[3] Bak, P., Tang, C.,& Wiesenfeld, K. (1988). Self-organized criticality. *Physical Review A, 38*, 364-374.

[4] Bak, P., Tang, C., & Wiesenfeld, K. (1987). Self-organized criticality: an explanation of 1/f noise. *Physical Review Letters, 59*, 381-384.

[5] Bettinger, P., & Zhu, J. (2006). A new heuristic for solving spatially constrained forest planning problems based on mitigation of infeasibilities radiating outward from a forced choice. *Silva Fennica, 40* (2), 315-333.

[6] Boettcher, S. (2000). Extremal Optimization: Heuristics via Co-Evolutionary Avalanches. *Computing in Science and Engineering, 2* (6), 75-82.

[7] Boettcher, S., & Percus, A. G. (2001). Extremal Optimization for Graph Partitioning. *Physical Review E, 64*, 026114.

[8] Boettcher, S., & Percus, A. G. (2003). Extremal Optimization: An Evolutionary Local-Search Algorithm. *Proceedings of the 8th INFORMS Computing Society Conference*.

[9] Boettcher, S., & Percus, A. G. (1999). Extremal Optimization: Methods derived from Co-Evolution. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 825-832). San Francisco, CA: Morgan Kaufmann.

[10] Casella, D., & Potter, W. D. (2005). Using Evolutionary Techniques to Hunt for Snakes and Coils. *Proceedings of the 2005 IEEE Congress on Evolutionary Computing.*

[11] Chen, M.-R., Li, X., Zhang, X., & Lu, Y.-Z. (2009). A novel particle swarm optimizer hybridized with extremal optimization. *Applied Soft Computing.*

[12] Chen, M.-R., Lu, Y.-Z., & Yang, G. (2006). Population-based extremal optimization with adaptive Lévy mutation for constrained optimization. *Proceedings of 2006 International Conference on Computational Intelligence and Security (CIS'06)* (pp. 258-261). Guangzhou, China: Springer Berlin.

[13] de Sousa, F. L., Ramos, F. M., Paglione, P., et al. (2003). New stochastic algorithm for design optimization. *AIAA J, 41* (9), 1808-1818.

[14] Diaz-Gomez, P., & Hougen, D. (2006). Genetic algorithms for hunting snakes in hypercubes: fitness function analysis and open questions. *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2006)* (p. 3890394). Las Vegas: IEEE Computer Society.

[15] Diaz-Gomez, P., & Hougen, D. (2006). The snake in the box problem: mathematical conjecture and a genetic algorithm approach. *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 1409-1410). Seattle: ACM Press.

[16] Eldredge, N., & Gould, S. J. (1972). Punctuated equilibria: an alternative to phyletic gradualism. In T. J. Schopf, *Models in Paleobiology* (pp. 82-115). San Francisco: Freeman, Cooper and Co.

[17] Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence.* New York: Wiley and Sons.

[18] Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

[19] Katz, J. I. (1986). A Model of Propagating Brittle Failure in Heterogeneous Media. *Journal of Geophysical Research*, 10412-10420.

[20] Kautz, W. (1958). Unit-Distance Error-Checking Codes. *IRE Transactions on Electronic Computers*, 179-180.

[21] Kennedy, J., & Eberhart, R. C. (2001). *Swarm Intelligence*. San Francisco: Morgan Kaufmann.

[22] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942-1948). IEEE Service Center.

[23] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*, 671-680.

[24] Leipins, G. E., & Potter, W. D. (1991). A Genetic Algorithm Approach to Multiple Fault Diagnosis. In L. Davis, *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

[25] Martin, M., Drucker, E., & Potter, W. D. (2008). "GA, EO, and DPSO applied to the discrete network configuration problem. *Proceedings of the International Conference on Genetic and Evolutionary Methods, GEM 2008*, 129-134.

[26] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics, 21*, 1087-1092.

[27] Middleton, A. A. (2004). Improved extremal optimization for the Ising spin glass. *Physical Review E, 69*, 055701.

[28] Moser, I., & Hendtlass, T. (2007). A simple and efficient multi-component algorithm for solving dynamic function optimization problems. *2007 IEEE Congress on Evolutionary Computation (CEC 2007)* (pp. 252-259). Singapore: IEEE.

[29] Peng, Y., & Reggia, J. A. (1987). A probabilistic causal model for diagnostic problem solving, part I: integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man, and Cybernetics, 17* (2), 146-162.

[30] Peng, Y., & Reggia, J. A. (1987). A probabilistic causal model for diagnostic problem solving, part II: diagnostic strategy. *IEEE Transactions on Systems, Man, and Cybernetics, 17* (3), 395-406.

[31] Potter, W. D., Drucker, E., Bettinger, P., Maier, F., Martin, M., Luper, D., et al. (2009). Diagnosis Configuration, Planning and Pathfinding: Experiments in Nature-Inspired Optimization. In R. Chiong, *Natural Intelligence for Scheduling, Planning, and Packing Problems.* Springer-Verlag.

[32] Potter, W. D., Miller, J. A., Tonn, B. E., Gandham, R. V., & Lapena, C. N. (1992). Improving the reliability of heuristic multiple fault diagnosis via the environmental conditioning operator. *Applied Intelligence, 2* (1), 5-23.

[33] Potter, W. D., Pitts, R., Gillis, P., Young, J., & Caramadre, J. (1992). IDA-NET: An intelligent Decision Aid for Battlefield Communications Network Configuration. *Proceedings of the Eighth IEEE Conference on Artificial Intelligence Applications (CAIA '92)* (pp. 247-253), March.

[34] Potter, W. D., Robinson, R., Miller, J., & Kochut, K. (1994). Using the Genetic Algorithm to find Snake-in-the-Box Codes. *Proceedings of the 7th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, (pp. 307-314).

[35] Tuohy, D., Potter, W., & Casella, D. (2007). Searching for Snake-in-the-Box Codes with Evolved Pruning Models. *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods* (pp. 3-9). Las Vegas: CSREA Press.

[36] Zhu, J., Bettinger, P., & Li, R. (2007). Additional insight into the performance of a new heuristic for solving spatially constrained forest planning problems. *Silva Fennica, 41* (4), 687-698.

[37] Zhou, T., Bai, W., Cheng, L., & Wang, B. (2004). Continuous extremal optimization for Lennard-Jones Clusters. *Physical Review E, 72*, 016702.