Diagnosis, Configuration, Planning, and Pathfinding: Experiments in Nature-Inspired Optimization

W.D. Potter, E. Drucker, P. Bettinger, F. Maier, M. Martin, D. Luper, M. Watkinson, G. Handy, and C. Hayes

Abstract We present experimental results of applying various nature-inspired optimization techniques to real-world problems from the areas of diagnosis, configuration, planning, and pathfinding. The optimization techniques we investigate include the traditional Genetic Algorithm (GA), discrete (binary and integer-based) Particle Swarm Optimization (DPSO), relatively new Extremal Optimization (EO), and recently developed Raindrop Optimization (RO); all inspired by different aspects of the natural world. We present algorithm setup, issues with adapting the various optimization methods to the selected problems, and the emerging results produced by the methods. We consider the GA to be the baseline technique because of its robustness and widespread application. The major contribution of this chapter deals with the fact that DPSO, EO, and RO have never been applied to the majority of these selected problems, making this the first time most of these results have appeared in the literature.

1 Introduction

A given search algorithm might perform exceptionally well for certain optimization problems while relatively poorly for others, and it is often quite difficult to explain why this is the case. It is therefore difficult to know beforehand which algorithms should be applied to which problems. In this chapter we investigate a number of nature-inspired optimization techniques and present preliminary experimental results of applying these techniques to real-world problems from the areas of diagnosis, configuration, planning, and pathfinding. The optimization methods we investigate include two population-based search schemes: the Genetic Algorithm (GA) and Discrete Particle Swarm Optimization (DPSO); and two single-element search

W.D. Potter
Institute for Artificial Intelligence, The University of Georgia, Boyd GSRC, Room 111 Athens, GA 30602-7415, e-mail: potter@uga.edu

schemes: Raindrop Optimization (RO) and Extremal Optimization (EO). Results show a comparison among the methods based on quality of outcome.

For the diagnosis problem, we use an instance of multiple fault diagnosis where we have 25 diseases each with 10 symptoms. The fitness of any single diagnosis is based on how well the diseases in the diagnosis explain the symptoms presented by the patient, whether or not the diseases explain symptoms not exhibited by the patient, and favoring the more common diseases over more rare diseases. The goal is to find the diagnosis containing the fewest diseases that explains all and only the patient's symptoms. Known optimal results are available for comparison with search technique results.

For the configuration problem, we apply our optimization techniques to the now outdated U.S. Army Mobile Subscriber Equipment network configuration problem. Many years ago, the U.S. Army used Mobile Subscriber Equipment to provide primary and backup communications for large-scale military operations. Once the requirements for an operation were determined, mission planners would prepare a "shopping list" of communications components including type of component and quantity. Components were integrated according to component constraints, requirements support, and Army doctrine (connectivity constraints) to form the communications network to support the mission. Reliable configurations are available for comparison with optimization results.

For the planning problem, we address spatial forest planning, which attempts to optimize timber harvest yields by selecting and scheduling specific forest stands for harvesting. Difficulty arises from the adjacency restrictions and timing constraints associated with harvesting the stands. We use a management unit containing 73 stands representing a typical forest from the western United States. Target results used for comparing the optimization techniques are available.

A pathfinding (also known as route finding) problem we have been investigating for some time is called the Snake In The Box problem. The idea behind this problem is to find the longest path, called the Snake, which has certain characteristics in a special type of graph called a hypercube. An *n-dimensional* hypercube has 2^n vertices where each vertex is connected to n other vertices. Using binary numbers to represent the vertex numbers shows that adjacent vertices differ by only one bit (e.g., vertex 101 is connected to vertices 001, 111, and 100 in a 3-dimensional hypercube). We focus on an 8-dimensional hypercube (containing 256 vertices and 1024 edges). The special path constraints require that each vertex on the snake path be connected to a successor and predecessor snake vertex (except the tail and head vertices which each have only one adjacent snake vertex), and that no snake vertex be adjacent to any other snake vertices (i.e., no chords). The optimal solution or longest snake in dimension 8 has yet to be proven, but the current longest known snake has length 97.

Of the optimization techniques examined, discrete Particle Swarm Optimization and the Genetic Algorithm are the population based search heuristics. The convergence of DPSO is influenced by the overall movement of the swarm as well as relationships among swarm particles with respect to local and best experiences. The Genetic Algorithm uses evolution-inspired operators to move through the search

space. Genetic material from mating parents is passed on to the resulting offspring that are typically more effective at surviving the environmental pressure applied by the associated fitness function. Individuals more adept at handling the pressure tend to contribute more genetic material to future generations.

Extremal Optimization and Raindrop Optimization are relatively new optimization techniques that are not population based but rather start from single points in the search space and use unique mechanisms for moving about within the search space. Extremal optimization, which has roots in statistical physics, focuses on removing weak components of the solution rather than favoring good components. Raindrop Optimization mimics the radiating waves generated by falling raindrops in a pond. In search heuristic terms, a feasible solution is perturbed which may result in an infeasible solution. As the infeasible solution is fixed, movement away from the initial search space location occurs. Perturbation and fixing continues to radiate from the initial location.

In the next section, we present overviews of the problem areas in more detail: multiple fault diagnosis, network configuration, forest planning, and pathfinding. Sections 3 and 4 describe the population-based and single element search schemes, respectively. Included in each search scheme discussion are characteristics of the scheme and how they map to the specific problem areas. Section 5 presents the results from each search scheme when applied to each problem area. These results are compared and discussed in order to provide insights into the appropriateness of applying a particular search technique to a specified problem. We conclude with some general observations regarding the search techniques.

2 Problem Areas

2.1 Multiple Fault Diagnosis

In electronics as well as medicine and other domains, multiple fault diagnosis is the identification of a set of problems (diseases) that best explains some observed abnormal behavior (symptoms). This type of problem solving is commonly referred to as abductive inference, and automating this approach has been the focus of numerous research efforts (Liepins and Potter, 2004; Potter et al., 1992b). A closer look at multiple fault diagnosis reveals three major obstacles between a diagnostic problem and a "reasonable" automated solution: the number of possible diagnoses, the "goodness" of a diagnosis, and the search scheme used.

Our goal is to find the diagnosis that best explains the observed symptoms. This best explanation is determined and wholly dependent on the calculation of the goodness of a diagnosis.

Typically, diagnosis quality is based on the notion of parsimoniously covering a set of observable symptoms (Reggia, Nau and Wang, 1983), that is, finding a minimal set covering of diseases that explains a given set of symptoms. The model

we adopt is the probabilistic causal model (PCM), introduced by Peng and Reggia (Peng and Reggia, 1987a,b). The PCM integrates "symbolic cause-effect inference with numeric probabilistic inference" to solve multiple fault diagnosis problems.

In the PCM, a multiple fault diagnosis problem is characterized as a 4-tuple $\langle D, M, C, M^+ \rangle$, where:

- D is a finite nonempty set of disorders (i.e., diseases).
- M is a finite nonempty set of manifestations (i.e., symptoms).
- C is a relation, a subset of $D \times M$. This relation pairs diseases with associated symptoms such that (d,m) in C means that disease d may cause symptom m.
- M^+ is a subset of M that identifies the observed symptoms (the symptoms that the patient presents). Note that symptoms not identified in M^+ are assumed to be absent.

A diagnosis DI (a subset of D) identifies the diseases that are possibly responsible for the symptoms in M^+. This leads directly to a bit-string representation with one bit per disease which is either part of the diagnosis ("on") or not ("off"). Diagnosis DI covers M^+ if each of the individual symptoms in M^+ is associated with at least one of the diseases in DI as determined using C. As with M^+, diseases not identified in DI are assumed to be absent.

Associated with each disease in D is a prior probability p indicating how rare or common the disease is. Associated with each causal association in C is a causal strength c such that $0 \leq c \leq 1$ and represents how frequently a disease causes a symptom. Note that c is not equivalent to the conditional probability $P(m|d)$ used in earlier Bayesian approaches. The causal strength represents the conditional probability $P(d \textit{ causes } m|d)$, and does not change, that is, we may expect the frequency with which d causes m to remain stable. An additional assumption stipulates that no symptom may exist in M^+ unless it is actually caused by some disease in D.

Now, we have $|D|$ prior probabilities and $|D| \times |M|$ causal strengths. Using these values, Peng and Reggia derive a formula for calculating the "relative likelihood", denoted $L(DI, M^+)$, of a diagnosis DI given observable symptoms M^+. The likelihood is the product of three factors:

$$L(DI, M^+) = L1 \times L2 \times L3 \qquad (1)$$

where:

- $L1$ is the likelihood that diseases in DI cause the symptoms in M^+. For diagnoses that do not cover M^+, $L1$ evaluates to 0 thus forcing L to 0. Unfortunately, this denies any analysis of non-cover diagnoses.
- $L2$ is the likelihood that diseases in DI do not cause symptoms outside of M^+ (e.g., in $M - M^+$). That is, $L2$ gives us a measure based on the expected symptoms that are actually absent. Ideally, we prefer $L2$ values that are close to 1.
- $L3$ is the likelihood that a highly probable (very common) disease d contributes significantly in the overall likelihood of a diagnosis DI containing d.

To summarize, $L1$ forces L to focus in on only diagnoses that cover or explain all symptoms in M^+, $L2$ forces L to focus on "irredundant" and "relevant" covers, and $L3$ forces L to focus on more likely or common diseases than on rare or less likely diseases. Irredundant covers do not contain any excess diseases that could be removed and leave us with a smaller cover. Relevant covers (which contain the set of irredundant covers) ensure that only diseases associated with symptoms in M^+ are seriously considered.

One aspect of the MFD problem that makes it unique relative to the other problems discussed here is that we base the results on diagnosing all possible symptom sets, not just one patient's symptoms. Since we have 10 symptoms, that amounts to 1023 diagnosis trials per experiment, excluding the healthy case. The quality of the search technique when applied to the MFD problem is based on the number of diagnosis trials (of the 1023 total trials) in which the search technique found the optimal solution. For example, if 763 trials resulted in the optimal diagnosis out of the 1023 trials, then the reliability of the search technique was determined to be 763/1023 or 75% reliable.

2.2 Network Configuration

In the early 1990s, U.S. Army communications networks were configured using Mobile Subscriber Equipment (MSE), which provided communications support for a typical five-division corps in an area of up to 15,000 square miles (MSE , 1990). The communications system was characterized as an area-switched system that provided both voice and data communications. An MSE network supported radio subscribers using VHF radio links, and supported wire subscribers using hardwired links and connectivity to other NATO forces and to commercial networks.

The backbone of an MSE network was composed of up to 42 Node Centers. Attached to the backbone were various types of Extension Nodes, Control Centers, NATO Interfaces, and Remote Access Units. The differences between the various types of Extension Nodes were directly related to the number of wire subscribers that could be supported and the actual fashion in which they were supported. A Large Extension Node could support nearly 200 wire subscribers, for example. Radio subscribers were supported by Remote Access Units. Rounding out the completion of a network configuration were the other various components including NATO interface units and System Control Centers.

Configuring an MSE network amounted to determining which components to use for a specified mission, and how many of each component to use (Chang and Potter, 1995; Potter et al., 1992a). The mission requirements identified only the number of wire subscribers and radio subscribers to support using the network. Military mission planners would spend many hours developing various scenarios that incorporated all aspects of the mission. The communications aspect of a mission — the development of a battlefield communications network that supports both wire and radio telephone communications requirements — is a particularly difficult task due

to the numerous dimensions involved, including the types of components and their quantities. We focus our attention on the "shopping list" of components needed for the mission. The shopping list corresponds to a network configuration and is a sequence of integers representing the number of components needed for the mission, with each position in the list corresponding to a different component. Consequently, we have a discrete integer-sequence problem representation that can be used directly by each of our heuristic search schemes. Of course, the network must function properly, support the current mission, and satisfy other configuration constraints (such as constraints imposed by U.S. Army doctrine, the terrain, and certain limitations of the components, such as proper connectivity).

Configuring an optimal network from various differing components and component quantities is the type of problem that is affected by combinatorial explosion, as the number of network components increases, the number of possible network configurations increases drastically. Using a familiar example, N network components may be configured into $N!$ networks (having N components in each network). This situation worsens if we allow varying network sizes.

Because of the combinatorial explosion in the number of possible networks, configuration algorithms that are guaranteed to find the optimal network structure via an exhaustive or near exhaustive search are of little value when dealing with larger networks. Since a typical MSE five-division corps contains over 40 Node Centers (not to mention the other types of components), it is clear that we need good heuristic approaches to solve the MSE configuration problem.

2.3 Forest Planning

In the forest planning problem, the goal is to obtain an optimal timber harvest schedule for a multiple-stand management unit forest subject to certain constraints. Specifically, in a viable harvest schedule, each stand is harvested in at most one of n harvest periods, and no two geographically adjacent stands are harvested in the same period. Among other things, the constraint prevents possible erosion and/or esthetic problems (this is called the green-up issue). For our specific tests, a problem from Bettinger and Zhu (2006) was used. The objective was to find an optimal harvest schedule for a management unit of 73 stands. Optimality is measured according to the function:

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

where i is the harvest period, H_i the total timber harvested in period i across all stands, and T denotes a per period target harvest volume for the entire forest. The total harvest cycle consisted of 30 years, with periods spaced 10 years apart (for a total of $n = 3$ periods). The goal is to achieve an even-flow harvest as close as possible to the target volume T, which for this management unit was given as 34,467 MBF (thousand board feet). The most reasonable representation for a harvest schedule is

a 73-element integer array where each position corresponds to a stand and contains the stand's harvest period.

Forest planning problems that involve the location and timing of harvest units usually require integer decision variables and can be difficult combinatorial problems to solve. Interest in using these types of forest planning techniques has recently increased (Bettinger and Chung, 2004), motivated by voluntary and regulatory programs, and facilitated by advancements in computer technology (Bettinger and Sessions, 2003).

2.4 The Snake In The Box Problem

The snake-in-the-box problem is a difficult problem in mathematics and computer science that was first described by Kautz in the late 1950's (Kautz, 1958). Snake-in-the-box codes have many applications in electrical engineering, coding theory, and computer network topologies. Generally, the longer the snake for a given dimension, the more useful it is in these applications (Klee, 1970).

Hunting for snakes (achordal-induced paths in an n-dimensional hypercube) deals with finding the longest path, following the edges of the hypercube, that obeys certain constraints. First, every two nodes, or vertices, that are consecutive in the path must be adjacent to each other in the hypercube, and second, every two nodes that are not consecutive in the path must not be adjacent to each other in the hypercube. A third constraint, whether the first node in the path is adjacent or non-adjacent to the last node in the path, determines if the path is a 'coil' or a 'snake.' While coils have received the most attention in the literature (Harary, Hayes, and Wu, 1988), both snakes and coils have useful applications. For this chapter we deal primarily with searching for open paths, or snakes, where the start node is not adjacent to the end node.

An n-dimensional hypercube contains 2^n nodes that can be represented by the 2^n n-tuples of binary digits of the hypercube's Gray code. By labeling each node of the hypercube with its Gray code representation, adjacencies between nodes can be detected by the fact that their binary representations differ by exactly one coordinate. In Figure 1, a path through the three-dimensional hypercube is highlighted. This path, $\{0,1,3,7,6\}$ in integer representation and $\{000,001,011,111,110\}$ in binary representation, is one specific example of a longest-possible snake for a three-dimensional hypercube. The length of this snake is four.

Common representations for the snake path include the integer sequence of nodes on the path (node sequence), an integer sequence of bit positions changed on the path (transition sequence), and a bit string representation where each bit represents a hypercube node and the snake path is identified by "on" bits.

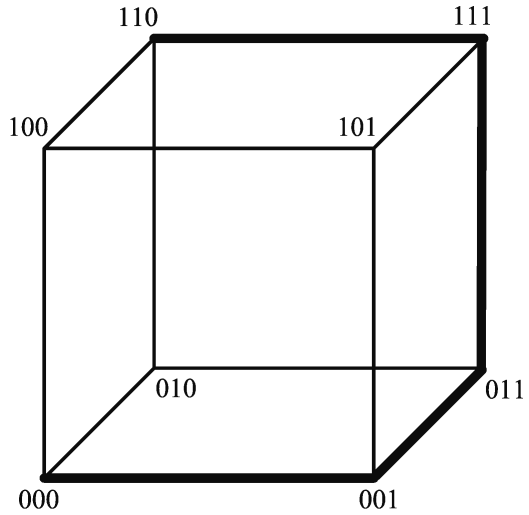

Fig. 1 A Three-Dimension Hypercube with Embedded Snake

3 Population-Based Search Schemes

3.1 The Genetic Algorithm

The Genetic Algorithm (GA) models the process of evolution by natural selection (Holland, 1975; Goldberg, 1989). The driving force of natural evolution is selection pressure, which is applied by competition between organisms for passing on their genetic material. In the GA, potential solutions are treated as individuals in a population, and pressure is exerted through the use of selection schemes that choose individuals for mating. A common mate selection scheme is k-ary tournament selection, where k individuals are selected at random and the competitor with the highest fitness goes on to mate. After two individuals are selected, they exchange genetic material through a process known as crossover. A very common example is two-point crossover, where two random points within the representation are selected and material between the points is swapped to create offspring with genetic material from the parents. Crossover combines elements of separate individuals to exploit good areas of the fitness landscape. The newly created solutions then undergo mutation, which randomly modifies a (randomly selected) component of an individual. Numerous variations exist for the GA. However, we try to stay with the Simple GA, which uses just the basic evolutionary operators described above.

Regardless of the flavor of GA used, the most important features by far affecting its performance are: (1) the way the problem is represented within the GA individuals, and (2) the fitness function, which is the mechanism used to compare the

strength of individuals. GA operators go hand in hand with the problem representation, and can lead to different levels of implementation difficulty. The problems addressed in this chapter lend themselves to different representation schemes. All but the Snake In The Box problem allow for straightforward representations and the use of standard operators. The SIB problem, however, has at least three quite different GA representations with corresponding repercussions on performance (e.g., the node sequence representation has adjacency issues, the transition sequence representation has length determination issues, and the bit string representation has both adjacency and length determination issues).

3.1.1 GA-MFD setup

In the early days of genetic algorithms, the primary representation scheme was a bit string corresponding to a solution in the problem search space. Multiple Fault Diagnosis lends itself directly to the bit string representation with each bit corresponding to a disease. The intuitive or natural interpretation of an individual bit string is a diagnosis where an "on" bit means the particular disease is included in the patient's diagnosis and an "off" bit means the particular disease is not present in the diagnosis.

For our MFD scenario, we incorporate a minor modification to the Relative Likelihood discussed earlier in order to allow it to work smoothly with the GA. Specifically, in the original relative likelihood scheme, two non-covering diagnoses would both evaluate to a zero fitness, whereas in the modified scheme it is possible for each diagnosis to have a very low yet distinct fitness value. This allows the GA some small measure of variance between non-covering diagnoses.

In addition to using the modified relative likelihood to compute the fitness, we have the casual association matrix which, based on practical knowledge of the domain, provides the measure of the relationship between the set of diseases and the set of symptoms. For our scenario, we have 25 diseases in the disease set, and 10 symptoms in the symptom set. Also, related to each disease is the notion of its occurrence within the population, that is, whether it is a common or rare disease. Consequently, each disease has associated with it a measure of a patient's likelihood of having the disease. During flu season, the probability that a patient has the flu is higher than the probability of the patient having Bubonic Plague, for example. Finally, we need to input the patient's symptoms before we can begin the search for the proper diagnosis.

As might be expected, reliability (i.e., the percent of optimal diagnostic trials) was generally better with larger population sizes (we typically used populations of 100, 200, and 300). Also, within a given population size, higher crossover rates tended to produce better reliabilities (we typically used 0.6, 0.7, and 0.8 for crossover probabilities). Finally, increased reliabilities were also observed with increased mutation rates within given population and crossover parameter settings (we typically used 0.005, 0.01, 0.015, and 0.02). Continuing to increase population size would allow us to continue to increase the diagnosis reliability but at the expense

of runtime. Continuing to increase either crossover or mutation probabilities led to decreases in reliability. The GA stopped each diagnostic search when either a total of 60 generations elapsed or there was no improvement for 20 consecutive generations (indicating a measure of convergence). In general, the GA performed fairly well giving reliabilities in the upper 80's (e.g., out of 1023 symptom sets, the GA found the optimal diagnosis 87% of the time).

3.1.2 GA-MSE setup

Configuring an MSE network involves determining the proper number of each type of equipment component to use for the given mission. We refer to the list of components and quantities as the mission "shopping list". The shopping list is represented by a sequence of integers corresponding to the number of components needed for the mission, with each position in the list corresponding to a different component. The set of available MSE components includes Node Centers (NCs), Large Extension Nodes (LENs), two types of Small Extension Nodes (SEN1s and SEN2s), System Control Centers (SCCs), NATO interface units (NAIs), and Radio Access Units (RAUs). The network must function properly, support the current mission, and satisfy other configuration constraints (such as constraints imposed by U.S. Army doctrine, the terrain, and certain limitations of the components, such as proper connectivity).

Each position in the sequence corresponds to a particular component type, and the values of the integers range between the minimum and maximum Corps complement for the component type. Because of the MSE representation, the typical selection and crossover operators within the genetic algorithm can be applied directly. However, the mutation operator was modified for integer sequences rather than binary sequences. In addition, in the scheme the mutation rate decreases over the evolutionary cycle allowing more exploration in the early generations and more exploitation in the later generations.

The MSE fitness function takes into consideration several important issues in determining the fitness of an individual (solution)(Potter et. al. 1992a). These issues include the mission requirements, the known support capacities of all the various components, the Division and Corps equipment complement, and the known minimum component requirements (such as, any network must have at least one System Control Center). In addition, issues such as the combined total number of components, and connectivity doctrine are included to help distinguish competing individual solutions. In the final analysis, the shopping list with the fewest components meeting the exact mission requirements, with the perfect component mix is the one we seek. The fitness of such a configuration would dominate configurations with less than perfect component mix, or those that did not exactly match the mission requirements (not too much support yet not too little support).

Our simple GA ran using the above representation and fitness function. We also used the age-proportional mutation operator, due to early trials showing the use of naïve mutation to perform more poorly. Trials indicated that crossover rate had

only a minor impact on results (in this case), so trials were run on population size (300, 400, 500) and mutation rate (5%, 10%, 15%), with 1,000 runs for each unique setting. The stopping criterion was ten generations with no improvement in the best fitness. The GA ran for an average of 34 generations and slightly less than 14,000 total fitness evaluations. In every case, increasing population size and mutation rate increased result quality, so only results using a mutation rate of 15% are discussed here. The most accurate trial was run with a population size of 500, giving near perfect results (the best solution was found 996 times in 1000 trials, 99.6% optimal), with the average number of fitness evaluations before finding the optimum at 12,000. When decreasing the population size to 400, the results quality was barely affected, dipping to 99.1% optimal, and the fitness evaluations for the best solution decreased to 9,200. When run with a population size of 300, results quality decreased to 98.1% optimal, with the GA finding the optimal solution after 6200 fitness evaluations.

3.1.3 GA-FP setup

We used an integer sequence representation for the Forest Planning problem to represent the harvest schedule. Within the western management unit we have 73 stands and each stand may be harvested during one of three time periods (or not harvested). Consequently, we used a 73-element (73-gene) representation where each position corresponds to a stand. The value of each element is an integer in the range of one to four indicating harvest time period.

The integer representation provides us with a natural harvest schedule interpretation. It also allows us to use the standard GA operators such as tournament selection and two-point crossover. Mutation is slightly modified in that when an element value is mutated, it is changed to another value within the proper range (similar to the mutation scheme used with the MSE problem).

We use the same fitness function defined in (Bettinger and Zhu, 2006) where a target time period harvest (34,467 MBF or thousand board feet) is specified for each time period, and we attempt to minimize the sum of the squared time period errors across the entire schedule. In addition, we experimented with introducing a penalty into the fitness function whenever an adjacency constraint was encountered. Recall, green-up and adjacency constraints may cause a harvest schedule to be invalid such as when two adjacent stands are scheduled to be harvested during the same time period. For the GA, we want the fitness to reflect the constraint violations but still provide some genetic material in order to allow the GA to progress through the evolutionary process.

We experimented with a variety of GA parameter settings but avoided using any "special" GA features such as seeding, or embedded local searching. We used typical GA parameter settings except for population size that was set somewhat high at 20,000 individuals. The crossover probability was set to 0.7 and the mutation probability was set to 0.17. Due to the complexity of the Forest Planning problem, the simple GA was only able to find a reasonably good harvest schedule but not as good as the target schedule reported by Bettinger. Their best harvest schedule pro-

duced a value of 5,500,391 whereas the GA derived schedule had a poorer value of 6,505,676.

3.1.4 GA-SIB setup

The bit string representation, used by Diaz-Gomez, illustrates the hypercube through 2^d spaces, with each space representing a node in the hypercube (Diaz-Gomez and Hougen, 2006a,b). A one-bit means that the associated node is turned on (included in the path) while a zero-bit means that the node is turned off (not included in the path). Even if the node is turned on, however, it does not mean that the node is in a valid snake path. In order to determine which nodes are included, the snake is evaluated with a length fitness function. Starting at node zero, the function uses the bit string representation to grow the snake outward by connecting valid nodes that are turned on. In order to confirm the validity of the snake, we used Diaz-Gomez's "vector of neighbors" matrix. The neighbors matrix is created by multiplying the individual snake bit string vector with the hypercube adjacency matrix. The results indicate how many of each node's adjacent nodes are on the snake path. The resulting matrix clearly represents the nodes that have more than two neighbors; thus violating one of the restraints of a valid snake.

The individuals created for the genetic algorithm are the simple bit strings containing 256 bits. Due to the extreme complexity of the SIB problem, for most of the trials we pre-set seeding the beginning portion of the individuals with the longest snakes found by Kochut (1996) for dimension seven (using various prefix subsequence lengths). In order to preserve the seed, we made sure that the seed was not touched by crossover or mutation. Even though the search space is exponentially large in dimension eight, we kept our population sizes below 1000 individuals at all times, with the best results occurring at population sizes of 100 and 350 individuals. Initially, roulette wheel selection was used to select parent individuals for reproduction. Later experiments utilized the more typical tournament selection. Four individuals were selected randomly for the tournament, and the individual with the highest fitness was chosen as a parent for the next generation. We used two-point crossover to exchange genetic material from two parents. Crossover did not occur all the time, however, but at different rates. Successful crossover rates varied from 40% to 80%. The new individuals, either with or without the occurrence of crossover, were then sent to a mutation function.

An adaptive mutation method was tried in order to maximize search space exploration. One version of the method started the mutation rate at 15%, with it decreasing with each generation, eventually approaching near zero after 250 generations. Failure to find a different longest length after four successive generations resulted in the mutation rate temporarily changing to 10%. Some experiments used a traditional mutation scheme, with the mutation rate fixed at 1% for the entirety of the program.

The most basic fitness function focused only on the length of the snake subsequences within the representation. Using the vector of neighbors, another fitness function incorporated lazy and isolated nodes (isolated nodes are nodes that are

turned on, but are not included in the snake path; and lazy nodes are nodes that are turned off and not adjacent to the snake path at all). Since the ultimate goal is to find the longest snake, we also incorporated skin nodes of a given snake (nodes adjacent to snake nodes but not on the snake). This relates to the tightness characteristic used by (Tuohy, Potter and Casella, 2007). Tightness reflects the idea of winding the snake through the hypercube in the tightest possible fashion. A characteristic of tightness is that individual skin nodes may be adjacent to many snake nodes; more means tighter. The function still focused on the length of the snake, but was also aided by how loose or tight the snake was wound through the hypercube.

With the bit string representation, we were able to find a snake of length 95. This came from experiments using 1000 individuals in a population, 1% traditional mutation scheme, 70% crossover rate, tournament selection, and pre-set seeding. We were routinely able to find mid-90's length snakes within 250 generations (a low number of generations), and on those occasions when the best lengths were lower, they never fell below length 85 snakes. Based on the computational resources used to process 100 individual populations for 250 generations, the mid-90's length result was outstanding considering that the longest known snake for dimension 8 has length 97!

3.2 Particle Swarm Optimization

Particle Swarm Optimization models the natural behavior exhibited by various large groups of animals such as flocks of birds or schools of fish (Engelbrecht, 2005; Kennedy and Eberhart, 1995, 2001). For birds, the movement of the flock is orchestrated by a combination of influences based on the speed and direction of the lead bird, and the speed and direction of an individual bird's immediate neighbors. Translating the metaphor to solutions within a search space of solutions, a small population of solutions can be "guided" towards the globally optimal solution based on the influence of: (1) the best member of the search space ever encountered, and (2) the "path" through the search space of each member of the population or swarm. The population "moves" in discrete time steps through the search space based on the following speed and direction characteristics for each member (Pugh and Martinoli, 2006):

1. The difference between the location of the best member and each member's location, ΔGB;
2. The difference between a member's best location and its current location, ΔPB;
3. A social influence: $C_1 \times Rnd \times \Delta PB$;
4. A cognitive influence: $C_2 \times Rnd \times \Delta GB$;
5. The new velocity, V_{new}: $inertia \times V_{old} + social\ influence + cognitive\ influence$, limited by V_{min} and V_{max}; and
6. The new location: $old\ location + int(V_{new})$.

Note that *Rnd* is a random number and that we make this standard PSO scheme a discrete scheme to match the problem representations we deal with (none of them lend themselves to the typical real valued PSO parameter setup). Based on the problem representation used, we either have a binary discrete PSO approach or an integer-based discrete PSO approach. In addition, the inertia or inertia weight is used to help balance the exploration and exploitation of the search.

3.2.1 DPSO-MFD setup

Our discrete PSO algorithm was implemented using Python and Microsoft's Visual Basic 6, and experiments were run using the same setup as the GA experiments (i.e., same bit string representation scheme for diagnoses and same modified relative likelihood fitness function). The representation of a diagnosis lends itself directly to the application of a binary discrete PSO approach (or simply a binary DPSO approach). Note that, to map the velocity values to binary values, we use the common sigmoid approach that compares the velocity to a random value, and assigns either zero or one based on the comparison result. Initial trials revealed that the performance of the algorithm was highly sensitive to inertia values. Inertia values below 0.7 and above 1.4 reduced the reliability significantly, and only inertia values of between 0.9 and 1.2 seemed to have any promise. Recall that with MFD, reliability is the ratio of optimal diagnosis sessions to total diagnosis sessions (for all possible symptom sets). Settings for the social and cognitive parameters did not have such a drastic effect. Values for these variables between one and four were tested during our initial trials. Setting the values between 1.9 and 2.2 gave the most promising results. From these initial trials, additional experiments were done to determine our best parameter settings for our binary DPSO and compare these results to other search schemes. We experimented with a number of different other parameters such as swarm sizes in the range of 100 to 500, and total generations in the range of 50 to 100 with stable generations set to 10.

Not surprisingly, our binary DPSO results were very good and very quick. We routinely achieved reliabilities in the upper-90's with modest DPSO parameter setting. For example, with a swarm size of 200, inertia set to 1.1, social value set to 2.2, cognitive value set to 1.9, limited to 60 generations, and with a stopping criterion of 10 stable generations, we were able to achieve 98% reliability (of the 1023 total symptom sets, we found the optimal diagnosis for 1002 of the symptom sets).

3.2.2 DPSO-MSE setup

The integer-based DPSO-MSE setup followed the GA-MSE setup where we used an integer sequence to represent the network configuration as well as the fitness function based on connectivity, mission support, and fewer network components. After several trial runs to narrow down good values for the social term (C_1) and cognitive term (C_2), the DPSO algorithm was run a thousand times for each of 12 different

parameter settings. The parameters included swarm size (100, 300); C_1 (1.5) and C_2 (4.0); inertia (0.4 to 0.9 with 0.1 steps); maximum allowable generations set to 300; a stopping criterion of no improvement for 20 consecutive generations; and V_{min} set to (-6), and V_{max} set to (6). With a swarm size of 300 individuals, C_1 set to 1.5, C_2 set to 4.0, and inertia varying from 0.4 to 0.9 (0.1 step size), we achieved 99.85% optimal results. That is, for six different inertia settings per run and each run repeated 1,000 times, DPSO found the optimal solution 5,991 out of 6,000 times (note that the second best solution was found three times and the third best solution was found twice). In the trials using a swarm size of 300, the average number of fitness evaluations was 31,500. Reducing the swarm size to 100 gave us a 98.63% optimal result. It was expected that the percent optimal would decrease, but surprisingly it only dipped slightly. This was achieved with C_1 set to 1.5, C_2 set to 4.0, and inertia varying from 0.4 to 0.9 for a total of 6,000 runs, of which we found the best solution in 5,918 runs (the second best solution was found 13 times and the third best solution was found only once). For these experiments, we found that DPSO required an average of 11,800 fitness evaluations per run to find the optimal solution.

3.2.3 DPSO-FP setup

Initially, the integer-based DPSO-FP setup seemed very straightforward and almost identical to the DPSO-MSE setup. We started with an integer representation with 73 elements corresponding to the 73 stands in the western management unit. Each element took on a value in the range of one to four, where four represented the situation when we did not harvest the corresponding stand. Many experiments were run in search of good DPSO parameter settings with social and cognitive values ranging from two to four, inertia ranging from 0.6 to 1.2 in 0.1 steps, and swarm sizes ranging from 1,000 to 10,000 in steps of 1,000. The DPSO was allowed to run up to 2,500 generations with a stopping criterion set at 150 stable generations. We experimented with the fitness function specified by Bettinger and Zhu (2006) as well as trying a fitness function containing a penalty value. The target harvest value to reach was in the 5.5M range.

The integer representation provided us with a natural harvest schedule interpretation and our early experiments utilized this representation extensively. Unfortunately, the results achieved using our integer-based discrete PSO search were poor. With a target of 5.5M, it was typically the case that the integer-based DPSO fitness would get only as low as about 150M (this seemed astonishingly high to us; recall we are minimizing). We also tried a functionally equivalent but implementationally different representation. Namely, we modified the representation from an integer string to a bit string where we used two bits for each of 73 stands (two bits are all that are needed to distinguish four items). Our discrete integer-based PSO was converted to a binary discrete PSO. The fitness function experiments stayed the same.

The binary DPSO experiments varied swarm sizes from 1,000 to 5,000 in steps of 1,000. The social and cognitive parameters, and inertia value varied as above from

two to four and 0.6 to 1.2, respectively. Generation and stable generation limits remained the same. Using this new setup, we were able to achieve fitness values in the neighborhood of 39M (much better than the 150M seen above) within 6M fitness evaluations but these results were still a long way for having the binary DPSO compete effectively with the other search schemes presented here. We tried a variety of other variations with the DPSO scheme in hopes of improving its performance with the Forest Planning problem but without success.

3.2.4 DPSO-SIB setup

In our discrete PSO implementation, the population is moved around the search space by adding the calculated velocity for each particle to its current value. The social and cognitive (C_1 and C_2) values that we used had C_2 set considerably lower than C_1. The reason for this was that we wanted the particle to be influenced by its own personal history more so than the global best because the global best may be a snake that is counterproductive to the individual being evaluated. There are many different, good paths through an eight dimensional hypercube and the global best may be a completely different path from our other individuals. Furthermore if the other individuals listen too closely to the global best, they could be influenced towards snake violations (with respect to where the individual currently is) that would worsen the individual to a point where it would not be able to recover. This is not to say the global best is not important, just that if an individual gives more weight to its own personal best, it is more likely to give good results because its personal best is attainable from where our individual's sequence of transitions places it in the search space.

Representation of individuals in the swarm is a key concern. In the DPSO-SIB case, the representation is an integer sequence where each position in the sequence contains a value representing the binary digit that changes when moving from one node of the hypercube to another. This is called a transition sequence. For example, the transitions followed when moving from node 0 to node 1 to node 3 and finally to node 7 are 1, 2, and 3, respectively (when identifying the node number using binary and counting from right to left starting from one). For these experiments, each individual had a length of 150. To check snake constraints, the transition sequences were converted into node sequences beginning with a common node (in this case, 0). Since every transition sequence is automatically a valid hypercube path, there was no need to check path validity for any potential solutions.

For the SIB problem, we also limited our velocity to the range -4 through 4 and tested inertia values of 0.3 to 1.2. We found that higher inertia values reduced the effectiveness of the DPSO for this problem. With no special DPSO-SIB setup features, we ran tests with the inertia set to 0.9 and the longest snake found was in the mid 60's. However, in most tests, the inertia was set much lower and the results were much better. Experiments were tried with swarm sizes from 1,000 individuals to 10,000 individuals. Typically, larger swarm sizes resulted in improved snake lengths.

To test for improved performance, we also used a slightly modified set of features for the DPSO-SIB, namely initial swarm pre-set seeding and a local growth scheme. This was necessary to help explore the very large search space (of which only a tiny portion contains valid snakes). The first technique consisted of seeding the initial swarm with fairly long snake subsequences. Since all transition sequences are valid paths, this was not necessary to the degree that it was with a node sequence representation, and only 5% of the swarm was seeded for these experiments. Using more seeds was stifling and inhibited the power of the search, yet using fewer seeds failed to produce good results. The initial seed came from dimension 7 and was of length 50. The other special feature introduced into our DPSO-SIB setup was a grow operator. This operator evaluated each individual in the swarm every generation, determined the location of its longest sub-snake, and then checked to see if any change could be made to the transitions adjacent to the sub-snake's ends in order to lengthen it. If so, the change was made to the individual. Overall, the longest snake the DPSO was able find was a snake of length 86 using relatively standard parameter settings and swarm seeding.

With the SIB problem, the goal is to find the longest snake. Although not reported here, we did run numerous experiments using the bit-string representation version of our DPSO. The results were disappointing. For example, using favorable parameter settings and allowing ample search time, the lengths of snakes found typically did not exceed the upper 30s. Achieving mid-80s with the transition sequence representation indicates that the problem representation is a very crucial aspect to consider for any knowledge engineer working on a difficult real-world problem.

4 Single Element Search Schemes

4.1 Raindrop optimization

The raindrop optimization search scheme or process is a recently developed local search technique (Bettinger and Zhu, 2006; Zhu, Bettinger and Li, 2007) that utilizes stochastic and deterministic methods to improve the quality of solutions. The method performs updates on a single solution yet may use a number of alterations to that solution prior to performing an update on the subsequent solution.

The raindrop optimization heuristic is performed on a single solution candidate S as shown below. It is set to run for N iterations:

1. Generate an initial solution S, and set S_{best} 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} 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.

The process is different from other heuristics in that it includes both stochastic and deterministic changes to a solution. Unlike many heuristics, the stochastic change is forced into the solution regardless of whether it results in constraint violations. In addition, this search process involves the use of a limited number of user-specified parameters: the number of iterations of the model, and the number of iterations X that pass before S reverts to S_{best} (Zhu, Bettinger and Li, 2007).

4.1.1 RO-MFD setup

We use a bit string representation and the relative likelihood fitness function for the RO-MFD setup (there is no need for a modified version since symptom non-covers are considered invalid diagnoses). We build an initial diagnosis that explains all the symptoms in the patient's symptom set (a super-cover) but not necessarily a maximal super-cover (all diseases associated with each patient symptom). When making a change to the diagnosis, we delete a disease and then, following the RO scheme, we check for constraint violations or, equivalently, a non-cover situation. Diseases are added back into the diagnosis based on an attempt to remain faithful to the notion of spatial closeness with RO as applied to Forest Planning. That is, we try to add a "close" disease back into the diagnosis that exposes the diagnosis to the fewest number of extra symptoms not in the patient's symptom set.

Experiments varied iterations and reversions, settling on 10,000 iterations and four reversions. Using more reversions intuitively seems to be attractive yet it allows the RO scheme to possibly drift away from quality solutions. Unfortunately, the RO scheme appears to be ill-suited for the MFD problem since we rarely achieved greater than 10% reliability (recall, reliability is a measure of goodness based on all possible symptom sets) and at the expense of over 10 million fitness evaluations. One observation during these experiments was the fact that the RO scheme works solely with valid diagnoses, ones that completely explain the patient's symptoms whereas the GA scheme may have actually produced diagnoses that were very close to but not covers. Our hypothesis was that using the modified relative likelihood might allow non-cover diagnoses to have higher fitness values than super-cover diagnoses. Fortunately, this turned out not to be the case but it leaves us still searching for the reason behind the poor performance.

4.1.2 RO-MSE setup

Network configurations are integer strings where each position corresponds to a specific MSE component. The fitness is based on connectivity, component requirements, and mission requirements. No real notion of spatial closeness exists within the MSE problem but other RO facets exist. The initial configuration was randomly determined and forced to meet the mission requirements. A configuration that did not support both the radio subscriber and wire subscriber requirements was identified as violating the mission constraints (similar to the adjacency and green-up Forest Planning constraints). Components were incremented and decremented during the iterations and reversions as long as the configuration remained valid.

Iterations varied from 5,000 to 20,000 with the optimal solution being found most of the time using 20,000 iterations. Reversions tended to follow the typical RO pattern where the best solutions were found using reversions set to either 4 or 5. The best single search resulted in the optimal configuration being found after just under 14,000 fitness evaluations. As for the frequency of finding the optimal configuration for the specified mission, our results were typically in the neighborhood of 72% optimal.

4.1.3 RO-FP setup

For the forest planning problem, the Raindrop optimization algorithm uses the fitness function and other data provided by (Bettinger and Zhu, 2006). It begins with a feasible harvest schedule, and in each iteration of the algorithm, the schedule is modified in the following fashion: A stand u is randomly chosen for modification and its treatment (i.e., whether it is cut at all, and if so in which period) is randomly modified. This modification might violate adjacency constraints. These are repaired by altering the treatments of each stand involved in the violation to restore consistency. Importantly, the stands are chosen for repair in order of increasing distance from u, and each repair must be consistent with all adjacent stands already repaired or else closer to u. The repairs thus radiate outward from u and continue until the schedule satisfies all constraints.

Our test essentially replicates the one found in (Bettinger and Zhu, 2006). The representation scheme is the same as that used by the GA. Specifically, harvest schedules were represented as a sequence of integers a_1, a_2, ... a_{73}, where each $a_i \in \{1,2,3,4\}$. The value for a_i denotes the period in which stand i is to be harvested (4 indicates that it is not to be harvested at all). For the test, data describing the forest — including adjacencies, x, y coordinates of centroids, and the timber volume obtained for each stand when harvested in a particular period — were stored in lookup tables.

In order to ensure that the algorithm yields increasingly improved schedules, the best schedule encountered thus far is stored. If after several iterations no improvement is found, the algorithm reverts to this best schedule. Following (Bettinger and Zhu, 2006), the reversion rate is set at four iterations. This is based on their experi-

ments with reversion rates ranging from 0 to 20. They conclude that a high reversion rate, which allows the algorithm more time to explore the search space, does not lead to better results. This is a conclusion we too arrived at when applying the RO search to our other problem areas.

The algorithm, implemented in Java, was run 50 times, with 100,000 iterations allowed per run. In each run, the initial schedule is simply a sequence of 4's (indicating no harvesting at all takes place). As shown in Table 1 below, the best result obtained was a schedule with fitness 5,500,391 (lower values are better). This matches the solution used as a baseline in (Bettinger and Zhu, 2006) and surpasses their raindrop result. We note that 100,000 iterations appears very low to us — it was chosen solely to replicate the experimental setup of (Bettinger and Zhu, 2006). As may be expected, increasing the iterations to 1,000,000 had the effect of producing the same harvest result very often.

Table 1 Target Values

Algorithm	Objective Function
Baseline Target	5,500,391
Raindrop (Bettinger and Zhu 2006)	5,556,343
Raindrop (replicated test)	5,500,391

4.1.4 RO-SIB setup

The raindrop optimization technique applied to the snake in the box problem produced the weakest result of all the search schemes. In an attempt to get better results, we tried using both a bit string representation and a transition sequence representation, but to no avail. Note that the transition sequence representation is the more conventional approach. For each representation, a slightly modified approach was taken for the application of raindrop optimization. Specifically the differences were with the correction of constraint violations due to the vastly different nature of the two representations. The bit string representation uses a binary array to represent the nodes of the hypercube that are "on" (in the path) and "off" (not in the path) rather than representing an actual path. This representation can make finding the longest path in the bit string difficult. For the transition sequence, each direction a path can travel from any node in the hypercube is assigned a numerical value based on the bit which changes, assuming each node is numbered in binary. The representation is then an array of values showing the direction to the next node of the path. For example in a 3 dimensional cube a path may be 1, 3, 2, and 3 (but this would not be a length 4 snake).

In the bit string representation, we can count the number of neighbors of a node that are "on" and store this in an array of neighbors following Diaz-Gomez. Then looking at the array, we can quickly see where the "heads" and "tails" of snakes are

by those entries with only one neighbor. Likewise, nodes with two neighbors are "body" nodes and any with three or more neighbors represent constraint "violation" nodes. By removing nodes that violate the longest snake in the current bit string solution, we expect the snake to grow longer. This process is the basis of the raindrop optimization applied to bit string snakes.

Starting with a random bit string with one bit for each node, a fitness function is used to locate the longest valid snake and its path is preserved. Next, a change is made by randomly turning on or off one or more nodes in the bit string, regardless of any violations to the best snake. Then we count and store neighbors, and begin correcting violations by turning off neighbors of nodes with three or more neighbors that are currently on. We make our corrections starting with nodes nearest to the changes we made and being sure to leave on nodes of the longest found snake. This continues until there are no nodes with more than two neighbor nodes turned on, radiating the changes outward from the change.

Following the radiating changes, the bit string contains either our original bit sequence snake, or numerous disjoint subsequences of the original snake. Thanks to the raindrop optimization approach, there should be no "violation" nodes. Running the fitness check once more, if the original snake has been improved and a longer snake was found, this longer snake is stored and the process repeats. If no better snake was found, we repeat the whole process for a set number of changes after which we revert back to the bit string with the best snake. Finally, we continue to repeat the process in the hope that a better snake may be found, as described in the raindrop description.

The best settings for the RO bit string representation were around 200 iterations with up to 128 changes and 51 repairs. With such a large number of changes before reversion, and the complexity of the fitness function, only rarely was a good snake found after the first 200 iterations. With these settings, we typically found snakes of length less than 55, and infrequently found snakes with lengths greater than 60, the best being 64.

For the transition sequence representation we first generate a random transition sequence and find the longest snake within it. This snake is then our best snake, and is stored as our valid transition sequence. Then we make one random change to the sequence by randomizing one of the directions taken, thereby twisting part of the snake within the hypercube. This twisting will almost certainly cause at least one constraint violation, so we must make additional changes to the snake. Starting with violations closest to the original transition, we change the direction taken at the node before the constraint violation occurs, continuing to change the transition sequence until no violations exist. Finally, we try to add one random direction to the end of the snake to improve its length (like a one-step grow operation). The rest of the methodology is the same as with the bit string representation, reverting to and storing better sequences as they are found.

Our transition sequence RO-SIB setup would not find snakes much longer than 40 and in effect shortened snakes more often than making them longer. We believe this to be due to the linear nature of the SIB problem, so we tried a minor modification to the RO scheme. Rather than correct closest constraint violations first, we

only corrected violations that came after the change, thereby helping to preserve the length of the snake found earlier in the transition sequence. Finally, to further aid preserving the snake, we implemented a weighting factor to determine where to make random changes to the transition sequence, favoring changes to transitions outside of the valid snake; this was based on an adjustable probability. With settings of 10,000 iterations with 51 changes and 12 repairs and a preservation probability of 10%, we found a snake of length 65 and on average produced snakes of length 59.

4.2 Extremal Optimization

EO is a recently developed local search heuristic (Boettcher and Percus, 1999) based on the Bak-Sneppen (BS) model of biological evolution (Bak and Sneppen, 1993). The BS model demonstrates self-organized criticality, a tendency for systems in statistical physics to organize themselves into non-equilibrium states. Unlike Darwinian evolution, the BS model addresses the evolution of entire ecosystems, called "neighborhoods", instead of individual species. These neighborhoods are made up of several species, each demonstrating relationships with its neighboring species. Thus, when the fitness of a species changes, the fitness of each one of its neighbors has the potential to change as well. Mutation is simulated by replacing the fitness of the least fit species in the neighborhood with a new random fitness. This affects the fitness of its neighbors, and causes a chain reaction through the entire neighborhood (Bak and Sneppen, 1993). EO individuals are based on these neighborhoods, and each component of the individual represents one of these species. Hence, unlike most optimization techniques, EO focuses on removing poor components of solutions instead of favoring the good ones.

The EO heuristic is performed on a single solution candidate S as follows:

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 the worst to be updated,
4. Assign the selected component a new value with a random fitness 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 a predetermined number of times.

Unfortunately, in some domains, step 3 can cause the heuristic to become stuck in local optima. To deal with this, Boettcher and Percus introduced a power-law distribution approach for selecting the component to be modified (Boettcher and Percus, 1999, 2001). To select a component using this approach, an integer $1 \leq k \leq N$ is determined from the probability function $P(k) \propto k^{-\tau}$, 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 (Boettcher and Percus, 2001). 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. Because of this stochastic nature, EO can jump in and out of near-optimal solutions at any point in time.

Another issue related to using 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 (de Sousa, Ramos, Paglione, and Girardi, 2003). 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.

Because of the stochasticity associated with higher values of τ, EO is likely to find an optimal (or near optimal) solution to a problem given adequate time. Unfortunately, problems with extraordinarily large search spaces may require infeasible amounts of time to find these quality solutions. Taking this into account, we created an efficiency measure to calculate how well the heuristic performs with different parameter settings. After performing several runs, an efficiency measure can be calculated by dividing the percentage of optimal solutions found by the number of iterations used as the stop criterion. For example, consider an experiment that found the optimal solution 75 percent of the time after 50 iterations and found the optimal solution 100 percent of the time after 500 iterations. The efficiency measures would be 1.5 and 0.2 respectively, thus demonstrating that even though we can find the optimal solution 100 percent of the time, we might be better off using the parameter settings that only found it 75 percent of the time and running the heuristic multiple times.

4.2.1 EO-MFD Setup

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 within the EO section) 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.

4.2.2 EO-MSE Setup

For the MSE problem, initial trials using an evaluation function that assigned a normalized weight to each component in the configuration gave poor results. To fix this, a "direction-based" scheme (Martin, Drucker, and Potter, 2008) was included. For each component in the configuration, the direction-based scheme evaluates the fitness twice, once with a decreased component value and once with a higher component value, keeping each component within Corps bounds. The components are then assigned directions of -1, 0, or 1. This corresponds to the component value that is less than, equal to, or greater than the current value and that results in a higher fitness. The EO search is carried out normally, but when a component is selected, it is modified only in the direction provided. In cases where a direction of zero is assigned, selected components are assigned a new random value. This helps to preserve the stochastic variability of the EO algorithm.

As with other EO setups, tests were performed to find the highest efficiency measure and the highest number of optimal solutions over repeated runs. For both tests, direction-based EO was used with τ in the range of $4 \leq \tau \leq 6$ (in increments of 0.01), and the range of total EO iterations was 100 to 20,000 (in increments of 100). At its most efficient point, our EO algorithm found only 27.8% optimal solutions, using 1400 iterations and $\tau = 5$. We were able to achieve perfect results over repeated runs, 100% optimal solutions, using 19,400 iterations and $\tau = 5$. For comparison, the highest efficiency measure found was 0.0199, but for the most optimal solutions, the efficiency was 0.0051. An interesting note is that even limiting the algorithm to 10,000 iterations still led to finding optimal solutions 93.3% of the time.

4.2.3 EO-FP Setup

During our 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 Forest Planning 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 (with the best known produced by our RO-FP setup experiments) are available. 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 $\tau = 1.5$.

4.2.4 EO-SIB Setup

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 similar to our GA-SIB setup. 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 similar to some of our other setups.

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 Results and Observations

In this section, we bring all of the best results together to show the effectiveness of the search schemes used in the experiments (see Table 2 below). Some expected results turned up but some unexpected results turned up as well. Some of the search schemes are in some ways similar to each other — the GA and DPSO search schemes both use populations of individuals to explore the search space, while the

RO and EO schemes use single individuals. Similarities among the problems also existed but only with respect to their corresponding search scheme implementation. The MSE, FP, and SIB problems naturally lent themselves to integer sequence representations, while at the same time the MFD, FP, and SIB problems had equally plausible bit string representations. Fitness functions were consistent across problems except for the RO-MFD setup that used the relative likelihood measure instead of the modified relative likelihood (an issue related to diagnoses that did not explain or over explained the patient's symptoms) because of the nature of the raindrop method.

Table 2 Results

	GA Search	DPSO Search	RO Search	EO Search
MFD	87%	98%	12%	100%
MSE	99.6%	99.85%	72%	100%
FP	6,505,676	35M	5,500,391	10M
SIB	95	86	65	74

After good parameter settings were identified, all of the search techniques produced fairly good results, but each technique also had an Achilles' heel. The GA and DPSO, the population based schemes, both exhibited strong MSE and SIB results, mixed MFD results, but surprising FP results (the performance of the particle swarm algorithm is especially surprising). Both the GA and DPSO schemes achieved remarkable MSE results — achieving greater than 99% optimal, indicating that these techniques, when applied to the MSE problem, can consistently produce excellent results. We were pleased to see that the binary DPSO search also performed very well with the MFD problem (98% of the 1023 symptom sets resulted in optional diagnoses). This is the type of reliability that gives us confidence in the appropriateness of this method. Although the GA reliability reached only 87%, this is actually quite good, especially since we restricted the GA-MFD to only simple features (e.g., no embedded local search was performed, and no fancy adaptive operators were used).

With the SIB problem, both the GA and DPSO techniques again achieved excellent results. The SIB problem has by far the largest search space of any of the problems addressed here. The DPSO technique's result of a snake of length 86 is quite good, particularly for a search technique that samples such a relatively small portion of the space. We were especially pleased with the GA's performance on the SIB problem since the known best length is currently 97. Much of our previous research into the SIB problem has been with the node sequence representation or the transition sequence representation. The use of the bit string representation—which led to such good results—has been very little researched. This is very encouraging and has given us a good deal of motivation to pursue this line of research more extensively.

As for the FP problem, the GA worked fairly well, producing the second best result found, but the DPSO scheme performed well below what we expected it to. This was very discouraging, and we are still uncertain as to the reason for such disastrous results. Much more analysis of the FP problem and tailoring of the DPSO to it is in order; this is high on our list of further research. Particularly, further research into more specialized operators and strategies, such as age-proportional mutation and embedded local search, could allow the population based schemes to perform even better.

The single element techniques also performed well in some cases, while in others they exhibited sub par or otherwise interesting behavior. Clearly, the RO search was expected to perform well on the FP problem, since its origins stem from the forest planning domain and it was tailor-made for problems in that domain. Consequently, it was no surprise that our implementation of RO found the best known solution in short order. What is surprising is that RO had some difficulty with the other problem areas we tested. For example, finding the optimal diagnosis in only approximately 1-in-10 diagnosis sessions is unacceptable MFD performance. The cause is still under investigation but surely it is related to the notion of "spatial closeness", a concept well defined in the forest planning problem but which does not carry over perfectly well to multiple fault diagnosis. The notion of spatial closeness surely hindered the RO technique when applied to the SIB problem as well. We consider a snake of length 65 to be an average length for a search scheme to locate. Note that EO was the best performer when applied to the MFD problem.

The results achieved by the RO algorithm for the MSE problem were satisfactory but not nearly as good as the results from the GA and DPSO schemes. However, the main issue with the RO-MSE results deals with the fact that when RO did not find the optimal solution, no clear pattern was displayed. That is, typically when the GA or DPSO schemes did not find the optimal MSE solution, they did find either the second best or third best solution (a common pattern with these population based schemes). However, non-optimal RO-MSE results exhibited no such similarity, but rather showed quite a bit of diversity; the results ranged from extremely poor to near optimal. Again, EO was the best performer when applied to the MSE problem.

The performance of EO is in stark contrast to that of RO. For the MFD and MSE problems, EO had a perfect record of performance, with perfect reliability for MFD and a perfect results for MSE. It also performed acceptably well with the SIB problem, even though it found the second shortest snake among all of the search techniques. A snake of length 74 is nevertheless a quality result for a single element search. Clearly, however, the GA was the best "snake hunter". The EO technique did have a little trouble with the forest planning problem. We continue to investigate methods to adapt it, just as we do to improve the results obtained in the DPSO-FP and RO-MFD problems.

In summary, all the techniques worked very well with the MSE problem although the RO scheme was not really in the same league with the GA, DPSO, and EO. The population-based schemes (GA and DPSO) did much better on the SIB problem than the single element schemes (RO and EO). The GA was competitive with the RO technique when applied to the forest planning problem; a problem for which RO was

custom made. EO did reasonably well with FP but DPSO performed surprisingly poorly. Interesting results came from the MFD problem area where we expected all of the techniques to do very well. Only DPSO and EO gave excellent results while the GA gave good results. The RO technique did not appear to mesh well with the MFD problem. The "silver lining" around our unexpected and unusual results is that these now represent opportunities for further research into these techniques and problems.

References

Bak P, Sneppen K (1993) Punctuated equilibrium and criticality in a simple model of evolution. Phys Rev Lett 71(24):4083–4086

Bettinger P, Sessions J (2003) Spatial forest planning: to adopt, or not to adopt? J For 101(2):24-29

Bettinger P, Chung W (2004) The key literature of, and trends in, forest-level management planning in North America, 1950-2001. Int For Rev 6:40-50

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

Boettcher S, Percus AG (1999) Extremal optimization: methods derived from coevolution. In: GECCO-99: Proc Genet and Evol Comput Conf. Morgan Kaufmann, San Francisco, pp 825-832

Boettcher S, Percus AG (2001) Extremal optimization for graph partitioning. Phys Rev E 64:026114

Chang FL, Potter WD (1995) A genetic algorithm approach to solving the battlefield communication network configuration problem. In: Yfantis, EA (ed) Intell Sys Third Golden West Intern Conf (Theory and Decision Library D, Vol 15). Kluwer, Dordrecht

Diaz-Gomez P, Hougen D (2006a) Genetic algorithms for hunting snakes in hypercubes: fitness function analysis and open questions. In: Seventh ACIS Intern Conf on Softw Eng, Artif Intell, Netw, and Parallel/Distrib Comput (SNPD'06). IEEE Computer Society, Los Alamitos CA, pp 389-394

Diaz-Gomez P, Hougen D (2006b) The snake in the box problem: mathematical conjecture and a genetic algorithm approach. In: Cattolico M (ed) Proc 8th annu conf on Genet and evol comput. ACM, pp 1409-1410

Engelbrecht AP (2005) Fundamentals of Computational Swarm Intelligence. Wiley and Sons, New York

Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Boston

Harary F, Hayes JP, Wu HJ (1988) A survey of the theory of hyper-cube graphs. Comput Math Appl 15:277-289

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Kautz WH (1958) Unit-distance error-checking codes. IRE Trans Electron Comp 7:179-180.

Klee V (1970) What is the maximum length of a d-dimensional snake? Am Math Mon 77: 63-65.

Kennedy J, Eberhart R (2001) Swarm intelligence. Morgan Kaufmann, San Francisco

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proc IEEE Intern Conf on Neural Netw. IEEE Service Center, Piscataway NJ, pp 1942-1948

Kochut KJ (1996) Snake-in-the-box codes for dimension 7. J Comb Math Comb Comput 20:175-185

Liepins GE, Potter WD (1991) A Genetic Algorithm Approach to Multiple Fault Diagnosis. In: Davis L (ed) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York

Martin M, Drucker E, Potter WD (2008) GA, EO, and DPSO applied to the discrete network configuration problem. In: Proc 2008 Intern Conf Genet and Evol Methods, GEM'08, CD Paper ID: GEM3397, pp 129-134

MSE (1990) Mobile Subscriber Equipment System: Reference Guide for the US Army. GTE Tactical Systems, Taunton, MA

Peng Y, Reggia JA (1987a) A probabilistic causal model for diagnostic problem solving, part I: integrating symbolic causal inference with numeric probabilistic inference. IEEE Trans Syst, Man, Cybern 17(2):146-162

Peng Y, Reggia JA (1987b) A probabilistic causal model for diagnostic problem solving, part II: diagnostic strategy. IEEE Trans Syst, Man, Cybern 17(3):395-406

Potter WD, Pitts R, Gillis P, et al (1992a) IDA-NET: an intelligent decision aid for battlefield communications network configuration. In: Proc 8^{th} IEEE Conf on Artif Intell Appl (CAIA'92). IEEE Computer Society, pp 247-253

Potter WD, Miller JA, Tonn BE, et al (1992b) Improving the reliability of heuristic multiple fault diagnosis via the environmental conditioning operator. Appl Intell 2:5-23

Pugh J, Martinoli A (2006) Discrete multi-valued particle swarm optimization. In: Proc 2006 IEEE Swarm Intell Symp, pp 103-110

Reggia JA, Nau D, Wang P (1983) Diagnostic expert systems based on a set covering model. Int J Man-Mach Stud 19(5):437-460

de Sousa FL, Ramos FM, Paglione P, et al (2003) New stochastic algorithm for design optimization. AIAA J 41(9):1808-1818

Tuohy DR, Potter WD, Casella DA (2007) Searching for snake-in-the-box codes with evolved pruning models. In: Arabnia HR, Yang JY, Yang MQ (eds) Proc 2007 Int Conf Genet and Evol Methods (GEM'2007). CSREA Press, pp 3-9

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