A COMPARISON OF NOVEL STOCHASTIC OPTMIZATION METHODS

by

PETER C. GEIGER

(Under the Direction of WALTER D. POTTER)

ABSTRACT

Many real world problems are too complex to solve with traditional programming methods in a reasonable amount of time. Stochastic optimization techniques have been applied to this class of problems with success. Set up and tuning an algorithm can be a daunting task, so this thesis first presents a method of simple optimization requiring no tuning parameters. Then, methods for dealing with search spaces with invalid solution space are introduced and compared.

INDEX WORDS: Optimization, Genetic Algorithm, Swarm Intelligence, Repair Operators

# A COMPARISON OF NOVEL STOCHASTIC OPTMIZATION METHODS

by

PETER C. GEIGER

BA, WARREN WILSON COLLEGE, 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

2015

A COMPARISON OF NOVEL STOCHASTIC OPTMIZATION METHODS

by

PETER C. GEIGER

| | |
|---|---|
| Major Professor: | Walter D. Potter |
| Committee: | Peter Bettinger |
| | Frederick Maier |

Electronic Version Approved:

Julie Coffield
Interim Dean of the Graduate School
The University of Georgia
May 2015

DEDICATION

This thesis is dedicated to my loving wife and friend through the end of the world:

Cody Luedtke

ACKNOWLEDGEMENTS

I would like to acknowledge the people who helped me get to this point. They were the people who wouldn't ever let me quit and always gave me encouragement. Chief DeSa, who told me that I should go get another degree and be a boss. Walter Potter who would not let me quit even when I tried. Lewis Fortner who gave me a scholarship and a stipend when I needed it the most. Finally, I'd like to acknowledge my mother and Will Richardson who listened to me drone on for hours about optimization.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

# CHAPTER 1

## INTRODUCTION AND LITERATURE REVIEW

Optimization is the process of making something as fully perfect as possible. Usually, in application, this means translating some desired outcome from the real world into a mathematical formula, which will then be maximized or minimized depending on the nature of the problem. Realistically the phrase "as fully perfect as possible," has a great impact on the work of optimization. A function's minimum may be near zero, but not quite, or conversely the maximum of a real world maximization problem may not be known. This leaves the developer of optimization algorithms with hard choices. Given infinite time, any complete search method will eventually find the optimal answer to a problem. However, the constraints of the real world are such that infinite time is not available and even when the search found the optimal answer, we would not be sure of the optimality until the complete search terminated.

Therefore, on truly hard problems (NP Hard) it is desirable to apply a method which yields good results in a relatively short period of time, reliably. The two papers presented in this thesis seek answers to some of the problems that arise when solving for the implementation problem above. In the first paper, Evolutionary Path Algorithm: A Simple and Extensible Metaheuristic for Global Optimization, an algorithm is presented that operates without any tuning parameters. It is compared across a range of applications and fairs reasonably well for a simple stochastic search method. It was an

experiment to develop an algorithm that could be implemented without as many unknown variables as the popular swarm and population based methods.

The second paper tackles a problem inherent in certain problems: areas within the search space that yield solutions that violate the acceptable terms of the search. There are several possible strategies for dealing with this problem: create an optimization technique that does not allow for invalid answers, allow the search to continue with invalid individuals in a population, or find a method to repair individuals efficiently so that they are once again viable. Two variations of this third option are compared in the paper.

## Stochastic Optimization Basics

The algorithms presented in this thesis are all non-deterministic optimization algorithms. In addition, most of them are also nature-inspired. These algorithms draw on metaphors from nature or physical systems to find the solution to problems that are too complex for a human to solve in a reasonable amount of time. The Genetic Algorithm developed by John Holland is the basis of much of this research and the inspiration for the algorithmic adaptations presented in Chapter 3 [1]. Swarm optimization techniques such as Particle Swarm Optimization [2] inspired the Evolutionary Path Algorithm presented in Chapter 2.

Stochastic optimization techniques all share a few things in common, so we will investigate the similarities between them first.

1. Problem
2. Objective function
3. Representation
4. Method for Manipulating Answers
5. Stopping Criteria
6. Problems Produced by These Structures

## Problem

Without a problem presented to develop a solution for, there is no need for an optimization technique. Many problems can be formulated for stochastic optimization, but they all have something in common. They must be able to be represented mathematically or by combinatorial abstraction. The knapsack problem is a classic example that illustrates this well. In this problem, an individual must fill a knapsack with a subset of items that maximize the value inside the knapsack without violating the carrying capacity of the bag [3]. The carrying capacity of the bag represents a constraint on the problem. Constraints create situations where the best answers are infeasible. In a world without constraints, we could just put all of the items available into our knapsack and very happily have all the value without a penalty. In the real world, problems have constraints that make them more difficult to solve.

## Objective Function

The objective function is borrowed from mathematical optimization and it represents a way to mathematically discern the fitness, or goodness, of a solution. Dantzig published the Simplex algorithm in 1947, which is an early algorithmic optimization solution that used the objective function in this way [4]. The English-readable explanation of the knapsack problem contains the objective function that we wish to express mathematically: fill a knapsack with a subset of items that maximize value without violating the bag's capacity. Mathematically we would express this in the following way: Maximize $\sum_{i=1}^{n} v_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0,1\}$ expressed in this way, we can allow the computer to evaluate many solutions very quickly.

## Representation

In early presentations of the Genetic Algorithm, Holland presents some interesting representations for the GA, some of which are very complicated [1]. However, simpler representations, such as binary arrays can abstractly represent many problems. This works well for certain problems, and is an obvious choice for the knapsack problem. For the knapsack problem, each of the *n* items can be arbitrarily assigned a number from zero to *n.* Then, when building a solution, we can put a 0 in the list if we do not wish to include an item, and a 1 if we do. This is a fairly efficient representation for this problem.

However, one can imagine a twist on the knapsack problem where this representation may not be the most efficient. Let us imagine that the number of items we have at our disposal is huge, in the millions, but the *variety* is rather limited, say fifty. We have many copies of a mere fifty items to work with, then the binary representation does not seem as efficient. In this case it seems to make sense to use an integer representation, where each of the fifty unique items has a place in an array and the total quantity each of those fifty is tallied and placed in the array. The Genetic Algorithm requires a slightly different crossover algorithm for working with arrays like this, but in general it remains the same.

Then there are problems that do not seem to work well with either of these representations. Problems of spatial dimension where the numbers exist on a continuous plane are a good example of problems of this class. Particle Swarm Optimization (PSO) is an optimization technique introduced by Kennedy and Eberhart in 1995 that mimics the social behavior of birds [2]. The representation of the problem in this instance includes not only the current solution, represented by planar coordinates, but also the

solution's velocity. This allows the particle to continue moving towards what may be the global optima. It is a very different encoding of information from the representations favored by the genetic algorithm, but it can be used to solve many of the same problems with just a bit of adaptation.

## Manipulation of Current Solutions

At the heart of all of the stochastic and nature inspired algorithms presented in this thesis is the way current solutions are manipulated to produce new solutions, which proceed in this way in an iterative fashion until some stopping criteria is met. In the Genetic Algorithm, the "genetic material" of two candidate solutions is mixed or "crossed over" in the hopes that the good parts of the solutions will be replicated to create a stronger solution. This action is explained by Holland's schema theorem, which says that short, low order schemata with above average fitness will increase exponentially in successive generations [1]. (Schema paper)

The PSO uses the tension between a particle's own observations and the observations of the swarm to locate good answers to hard problems. By not flying directly towards the best known point in the solution space, more space is explored and better answers are found [2]. The relationship between particles is important for finding good solutions. However, in both the Genetic Algorithm and the PSO, the settings that determine the behavior of the algorithm play an important role in the success or failure of the algorithms.

## Stopping Criteria

When to stop an algorithm is the final problem. Once an algorithm has converged, assuming it is a converging algorithm, it will likely not produce a better answer no matter how long you run it. Therefore, it is advantageous to run an algorithm

long enough to allow for convergence and no longer. There are ways to heuristically detect convergence, but experimentation is often more expedient than coding a solution to decide when an algorithm should stop optimization. In all of the experiments presented here a large number of iterations was chosen for the stopping criteria.

## Problems Addressed By the Algorithms Here:

Settings tuning represents a secondary problem beyond the optimization problem an algorithm is designed to solve. There are many papers written just to suggest good tuning parameters for a specific algorithm on specific problems. In the second chapter of this thesis, a method for optimization requiring very little tuning is presented.

The representation, objective function, and configuration of settings for these algorithms all play a major role in their success on any given problem. In a problem with constraints like: no answer may exceed the carrying capacity of our knapsack, the problem of violations can be handled almost anywhere in the structure of the algorithm. There may be a representation that prevents incorrect solutions from occurring, an objective function can be modified to penalize for constraint violations, or the algorithm itself may be modified in such a way that the new solutions it produces are valid solutions. An exploration of repairs is presented in the third chapter of this thesis because constrained problems make up a great deal of real world problems.

CHAPTER 2

# EVOLUTIONARY PATH ALGORITHM: A SIMPLE AND EXTENSIBLE METAHEURISTIC FOR GLOBAL OPTIMIZATION[1]

## Introduction

Many real world problems require combinatorial optimization. Examples include scheduling, bin packing, planning, network design, and engineering optimization. What all of these problems share is a search space that is generally too large for exhaustive search methods to find the optimal answer in a reasonable amount of time. Therefore, metaheuristic methods have proven to be useful because they can find good answers in a limited amount of time.

Metaheuristics employing diverse strategies have been developed over the past half century. The most notable being the Genetic Algorithm, Particle Swarm Optimization, Simulated Annealing, and various other swarm techniques such as Ant Colony Optimization [6,7,8,9,10]. These algorithms have produced diverse search strategies, operators for combining answers to form new ones, and new sets of rules for exploring the search space in an efficient manner. Although each of these metaheuristics searches in a different way, they all explore and exploit the search space to find good answers.

[1] Geiger, P.C., Potter, W.D., Richardson, W.D. 2013. *Proceedings of the ICAI.* Reprinted here with permission of the publisher.

The Evolutionary Path Algorithm explores and exploits the search space using a very simple greedy strategy and an exploitation, or intensification, operator called Path Relinking [5]. It does not however, inherit the algorithm as a whole, and instead combines random search with Path Relinking and a moderately greedy strategy. No gradients are computed and directionality is not required. The aim of the development of the algorithm was to make a metaheuristic that was simple to implement for any problem without an abundance of tuning parameters.

In the following we will present the Evolutionary Path Algorithm. It is an algorithm that traces the evolutionary links (differences) between potential solutions to find better solutions to hard problems. In section 2 we describe the Evolutionary Path Algorithm as applied to a minimal set cover problem. Section 3 describes the Evolutionary Path Algorithm as applied to a combinatorial optimization problem using integer values.

## EPA Applied to a Diagnosis Problem

### Evolutionary Path Algorithm

The Evolutionary Path Algorithm is most simply expressed by the following steps:

1. Initialize a Random Point within the Search Space. This is now the Candidate Solution and the Best Solution.
2. Initialize another Random Point within the Search Space. This is the Random Solution, if it is better than the Candidate Solution, it becomes the Best Solution.
3. An Evolutionary Path is built between the Candidate Solution and the Random Solution. Any solution found that is better than Best Solution replaces Best Solution.
4. Candidate Solution is replaced by Best Solution when the traversal of the Evolutionary Path is complete.
5. The process above is repeated until some stopping condition is met.

The Evolutionary Path Algorithm (EPA) is initialized by choosing a random point within the search space to become the candidate solution. Then, this solution is improved by choosing another random point within the search space and sampling the fitnesses of a randomly generated evolutionary path between the two points. If the best point along the path is better than the candidate solution, the old candidate solution is discarded in favor of the best point seen so far.

This process of traversing an evolutionary path to the best candidate solution is repeated until some stopping criteria, such as a number of fitness evaluations, or a number of paths traversed has been met. At the far end of the path the difference between our candidate solution and the sampled point may be high; this is where exploration of the search space occurs and there is a possibility of leaving local optima in favor of the global optima. As the difference between our candidate solution and points along the evolutionary path decreases, EPA is implicitly performing a local search with a chance of incrementally optimizing our candidate solution.

Creating an evolutionary path is very simple when locations within the search space are represented as binary strings. An XOR of the two strings identifies the bits for which the locations differ. The number of differences is known as the Hamming distance between the two bit strings. Then, one of these bits that differ is chosen at random. The value of the randomly chosen differing bit is then set to match that of the candidate solution. The fitness of this intermediate point along the evolutionary path is evaluated. If the fitness of this intermediate point is better than the fitness of our candidate solution, we remember its location as the “Best Solution”.

The evolutionary path is complete when there are no more intermediate points to sample. If, at the end of this process, a best location exists that is not the candidate solution, the candidate solution is replaced by the best solution seen thus far. This process of path building continues until some stopping criteria has been met.

Below is an example path that could be built between two bit strings with a Hamming distance of 3. Although there are multiple possible paths that *could* be built between most solutions, only a single path is traversed in the Evolutionary Path Algorithm.

**Step 1 – A Candidate Solution and Random Solution Pairing with Three Differing Bits.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 1 | 0 | 0 | 0 |
| Random Solution | 0 | 1 | 0 | 1 |

**Step 2 – The second bit of the Random Solution was chosen at random to build the second point in the path, reducing the Hamming distance to 2. If the new Random Solution is better than the Candidate Solution it is remembered as the Best Solution but the rest of the path is still built.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 1 | **0** | 0 | 0 |
| Random Solution | 0 | **0** | 0 | 1 |

**Step 3 – The first bit of the Random Solution was chosen at random and made to match the first bit of the Candidate Solution to build the third point in the path, reducing the differences between the solutions to 1.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | **1** | 0 | 0 | 0 |
| Random Solution | **1** | 0 | 0 | 1 |

**Step 4 – The Path is Complete. If no solution along the path was better than the Candidate solution, the algorithm begins again with the Candidate Solution as the Best Solution.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 1 | 0 | 0 | **0** |
| Random Solution | 1 | 0 | 0 | **0** |

## Multiple Fault Diagnosis

The Evolutionary Path Algorithm has been applied to the Multiple Fault Diagnosis problem (MFD) [12] for characterization. MFD is an NP hard minimum set cover problem in which diagnoses for a set of symptoms are represented by a binary string where an "on" bit represents that the disease is indicated by the symptoms and an "off" bit represents that the disease is not indicated by the set of symptoms. In the problem as presented, there are 10 possible symptoms or 2^10 - 1 (1,023) possible symptom sets. We did not consider the empty set of symptoms because it does not make sense to diagnose people who are well. For this set of symptoms there are 25 possible diseases with 2^25 minus 1 or (33,554,431) possible diagnoses. A set of observed symptoms ($M+$) is evaluated against a diagnosis (set of possible diseases) ($DI$) by the following fitness function:

$$L(DI;M+) = L1xL2xL3$$

The likelihood of any diagnosis (DI;M+) is L1xL2xL3 where

*L1* is the likelihood that diseases in *DI* cause the symptoms in *M+*.

*L2* is the likelihood that diseases in DI do not cause symptoms outside of M+.

*L3* is the likelihood that a highly probable (very common) disease *d* contributes significantly in the overall likelihood of a diagnosis *DI* containing *d*.

The values for calculating *L1*, *L2*, and *L3* are contained in tables mapping the likelihood of any disease causing a given symptom and the probability of any disease *d* actually occurring in the real world. For example, the table used in *L3* biases the problem towards the common diseases (perhaps the common cold) and away from obscure diseases like the bubonic plague. The effect of the fitness function is to favor diagnoses (*DI*) that explain all of the symptoms present (*L1*) without explaining extra symptoms (*L2*) and contain common diseases instead of rare (*L3*).

Since the search space is made up of possible diagnoses and these are already bit-strings, the set-up for the EPA is minimal. The fitness function, a function that builds evolutionary paths, and a reasonable stopping point are the only functions necessary to build the algorithm.

### Reliability of EPA for MFD

To test the algorithm's reliability it was run against all of the possible symptom set combinations, which is 2^10 - 1 (1023), ten times. The reliability for each run was then computed by dividing the number of times the algorithm found the optimal solution by 1023. For example, if the algorithm found the optimal solution 788 times out of 1023, it would have 77% reliability for that run. Because EPA is a stochastic algorithm, it was

run 10 times and the reliability for each of those 10 runs was averaged to determine its reliability.

For this test, the stopping condition was set to a maximum of 18,000 fitness evaluations per symptom set (M+). This number was chosen because it represents the maximum number of fitness evaluations Potter et al [12] allowed for the Genetic Algorithm to test reliability. The Genetic Algorithm represents the algorithm that is most similar to EPA out of those that Potter et al ran (Genetic Algorithm, Discrete PSO, Rain Drop Optimization, and Extremal Optimization). Because the optimal fitness values are known for this problem, the algorithm was stopped for each symptom set (M+) if the known best fitness was found before 18,000 fitness evaluations had been performed. The reliability statistics for other algorithms were provided by Potter et. al[12].

**Table 1. Comparison of Algorithm Reliabilities for MFD**

| **EPA** | **GA** | **DPSO** | **EO** |
|---|---|---|---|
| 97% | 85% | 98% | 100% |

EPA's reliability is comparable to the DPSO's reliability for this problem and is far better than the GA's with a similar number of fitness comparisons. Extremal Optimization has a reported 100% reliability on this problem with a similar number of evaluations as allowed. It is likely that EPA's reliability is higher than the GA's with a similar number of fitness evaluations due to the fact that the GA has many redundant fitness evaluations as it nears convergence whereas the EPA continually builds evolutionary paths to random possible solutions so the likelihood of redundancy is extremely small.

## EPA Efficiency for MFD

To test the efficiency of EPA on the MFD problem, the algorithm was run 10 times over all 1023 symptom sets. The stopping condition for these runs was the optimal solution *DI*, so the algorithm was forced to find the global optima for each symptom set. This experiment was done to replicate an earlier reliability experiment for several algorithms [12].

On average the EPA took 5971 fitness evaluations to find the optimal solution. Potter et. al. did not provide average fitness evaluations to find optimal solutions for the MFD for any of the algorithms compared for the MFD. However, 5971 fitness evaluations is considerably less than the maximum allowed by Potter et. al., because a GA with 300 individuals run for 60 generations allows for a maximum of 18,000 fitness evaluations per solution.

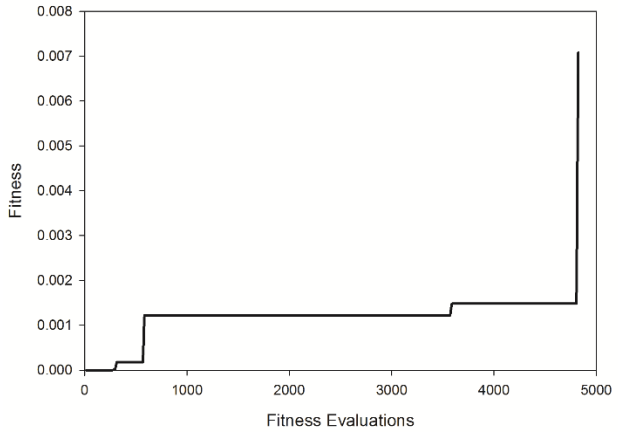


**Figure 1. A Selected Optimization Curve of EPA for MFD**

## EPA With Integer Combination

### Mobile Subscriber Equipment Problem

The mobile subscriber equipment problem (MSE) is a configuration problem in which the best mobile communications network must be assembled within the constraints of component connectivity, mission requirements, and Army doctrine. Although the networks being configured are now outdated, the MSE represents a classic configuration problem in which an expert's knowledge can be translated to a fitness function for optimization.

Army networks configured using MSE were in use in the early 1990s. A single network could support troops over a 15,000 square mile radius. At the heart of any MSE system was the Node Center. Each network could contain up to 42 Node Centers. There are six other modules that attach to this backbone to form the rest of the network. Both large and small extension nodes expanded the number of wire subscribers by big and small amounts respectively, with two types of small extension nodes offering different combinations of connectivity. Radio subscribers were supported by Remote Access Units, Control Centers were necessary for connectivity, NATO interface units allowed the system to be patched into NATO communications systems.

Using the fitness function laid out by Chang and Potter it is possible to optimize a network given the number of wire and radio subscribers [13]. However, the search space is very large due to the combinatorial explosion that is created. The minimum number of units allowed for most of the equipment is zero, with the exception of the Node Center and System Control Center. At least one of these is required for each network. The maximum number of allowed units ranges from the low end at 4 NATO Interfaces to 168 small extension nodes.

## EPA Set-Up for MSE

Building an evolutionary path between two lists of integer values is almost as simple as building a path between two bit strings. Originally, a naïve scheme of iteratively setting the values of the random sample to those of the candidate solution, in the same way that the bits were switched in the binary example, was tried. This did not yield positive results.

The second scheme worked in the following way:

While stopping criteria not met:

1. Select a Random Solution
2. While Path not Complete (Random and Candidate Equal):
    a. Find differences between the Random Solution and the Candidate Solution
    b. Select a differing component at random.
    c. Choose a random value between the Candidate solution's assignment for the differing component and the Random solution's assignment for differing component.
    d. Evaluate the new Random Solution: if this point has the highest fitness seen so far, save it as the Best solution.
3. Restart randomly if stagnation is detected.

An upper bound of 300,000 evaluations was used because early runs suggested that this represented the near upper bound for the number of evaluations the algorithm would take. However, the average number of evaluations needed to find the optimal answer was much lower than this. The fitness function needs a number of wired subscribers and a number of wireless subscribers to evaluate a candidate solution. The number of wired

subscribers was set to 1,495 and the number of wireless subscribers was set to 672. This combination has an optimum fitness of 327.35689.

Random restarts were also introduced due to the observed possibility of stagnation. For these experiments, stagnation was assumed if the algorithm had not found the optimal answer by 125,000 fitness evaluations.

Obviously building paths between solutions with very different component values can get quite long. Below is a sample path built between two solutions with just a single component that differs in value.

**Path Point 1 – a Candidate Solution and Random Solution Pairing with a Single Component that Differs.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 100 | 14 | **98** | 0 |
| Random Solution | 100 | 14 | **12** | 0 |

**Path Point 2 – Since the Third Component Differs, a random value between the Candidate's (98) and the Random's (12) value assignment is selected to replace the Random Solution's current value for the third component.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 100 | 14 | **98** | 0 |
| Random Solution | 100 | 14 | **72** | 0 |

**Path Point 3 – Random Solution's Third Component is Still not Equal to Candidate Solution's Third Component, so the process is repeated again.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 100 | 14 | **98** | 0 |
| Random Solution | 100 | 14 | **97** | 0 |

**Path Point 4 – The Two Solutions are Now Identical. If Any Point Along the Path Had a Higher Fitness Than The Candidate Solution, it Would Now Become the New Candidate Solution.**

| | | | | |
|---|---|---|---|---|
| Candidate Solution | 100 | 14 | **98** | 0 |
| Random Solution | 100 | 14 | **98** | 0 |

### Reliability of EPA for MSE

To test the reliability of EPA, the algorithm was run 1,000 times. The stopping condition for these 1,000 runs was 300,000 evaluations or the optimal solution. This experiment replicated a previously published reliability experiment for several other popular optimization algorithms run on the same problem [8]. For this experiment reliability was calculated as the number of times the algorithm found the optimal solution, divided by the number of runs.

**Table 2. Comparison of Algorithm Reliabilities for MSE**

| **EPA** | **GA** | **DPSO** | **EO** |
|---|---|---|---|
| 99.3% | 99.6% | 99.85% | 100% |

The EPA found the optimal solution 993 times out of 1000, missing the optimal solution just 7 times. In each of these 7 times a random restart was initiated, and the

second or third best solution was found. All of the algorithms had less than a 1% margin of difference in reliability for this problem. The real difference between the algorithms' performances was in efficiency.

### EPA Efficiency for MSE

To test the efficiency of the EPA for MSE the algorithm was run 1,000 times with the optimal solution as the only stopping condition. This setup was used to force the algorithm to find an optimum even in a badly performing run. The EPA took an average of 49,906.5 evaluations to find the optimum solution for this problem, with a minimum of 1,675. This is larger, but similar to the number of evaluations that the DPSO and GA took in reported results [12]. These are presented in the table below. There were no results for the number of fitness evaluations for EO.

**Table 3. Average Fitness Evaluations to Find Optimum**

| **EPA** | **GA** | **DPSO** |
|---|---|---|
| 49,906.5 | 12,000 | 31,500 |

The observed difference in efficiency for this problem is due to the observed possibility of spending a great deal of time on a good, but not globally optimum point. In later experiments, not presented here, random restarts were introduced when this situation was detected, which improved efficiency greatly for this problem. However, the basic algorithm remains less efficient than both GA and DPSO for this problem.

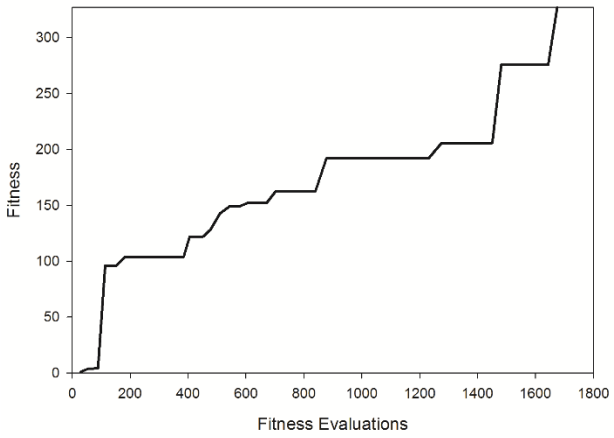


**Figure 2. A Selected Optimization Curve for MSE**

## Conclusions

This paper has presented a new optimization algorithm designed particularly for discrete optimization. The Evolutionary Path Algorithm requires no tuning parameters, but still performs well in comparison to other well-known stochastic optimization algorithms. It is reliable and reasonably efficient. Although the algorithm may not be as efficient on real-valued optimization problems as competing algorithms, it is likely that small modifications could greatly improve this performance.

The algorithm finds good solutions without calculating any gradient information, using tuning parameters, or keeping a large population in memory. Because it continually samples the whole search space it is able to continue to search for and find good solutions long after other methods have converged. Future work should extend the path-building operators to include other representations and look for extensions of the algorithm that tailor it to specific optimization problems.

## REFERENCES


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

[2] Kennedy, J. & Eberhart, R.C. 1995. Particle swarm optimization. Proc. IEEE International Conference on Neural Networks, Picataway NJ: IEEE Service Center, 1995, 1942-1948.

[3] Mathews, G. B. (25 June 1896). "On the partition of numbers". Proceedings of the London Mathematical Society 28: 486–490.

[4] Dantzig, G. 1963. *Linear Programming and Extensions*. Princeton University Press, West Sussex.

[5] F. Glover. "Tabu Search—part I." ORSA Journal On Computing , Issue 3, 190-206, 1997.

[6] S. Boettcher & A. G. Percus. " Extremal Optimization : Methods derived from co-evolution." Modeling and Simulation Design. AK Peters Ltd. 1999.

[7] S. Kirkpatrick, S.&Vecchi, M.C.. "Optimization by simmulated annealing." Science, 220(4598), 671-680, 1983.

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

[9] Kennedy, J. & Eberhart, R.C. 1995. Particle swarm optimization. *Proc. IEEE International Conference on Neural Networks*, Picataway NJ: IEEE Service Center, 1995, 1942-1948.

[10] D. T. Pham, E. Ghanbarzadeh, A. Koc, E. Otri, S. Rahim, S. &Zaidi, M. 2009. The Bees Algorithm – A Novel Tool for Complex Optimisation Problems.*Innovative Production Machines and Systems,2009.* *DOI=http://conference.iproms.org/the_bees_algorithm_a_novel_tool_for_complex_optimisation_problems.*

[11] Dorigo, M., & Di Caro, G. 1999. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (Vol. 2). IEEE.

[12] Potter, W. Drucker, E. Bettinger, P. Maier, F. Luper, D. Martin, M. Watkinson, M. Handuy, G. &Hayes, C. 2009. Diagnosis, Configuration, Planning, and Pathfinding: Experiments in Nature-Inspired Optimization.*Natural Intelligence for Scheduling, Planning and Packing Problems,* R. Chiong, ed., Springer-Verlag.

[13] FL Chang and W. Potter. A genetic algorithm approach to solving the battlefield communication network configuration problem. In: Yfantis, EA (ed) Intell Sys. Third

GoldenWest Intern Conf (Theory and Decision Library D, Vol 15). Kluwer,Dordrecht. 1995.

CHAPTER 3

# TWO REPAIR SCHEMES APPLIED TO A MULTIPLY-CONSTRAINED HARVEST SCHEDULING PROBLEM[2]

## Section 1.0 Introduction:

This paper compares two novel repair methods for optimizing timber harvest

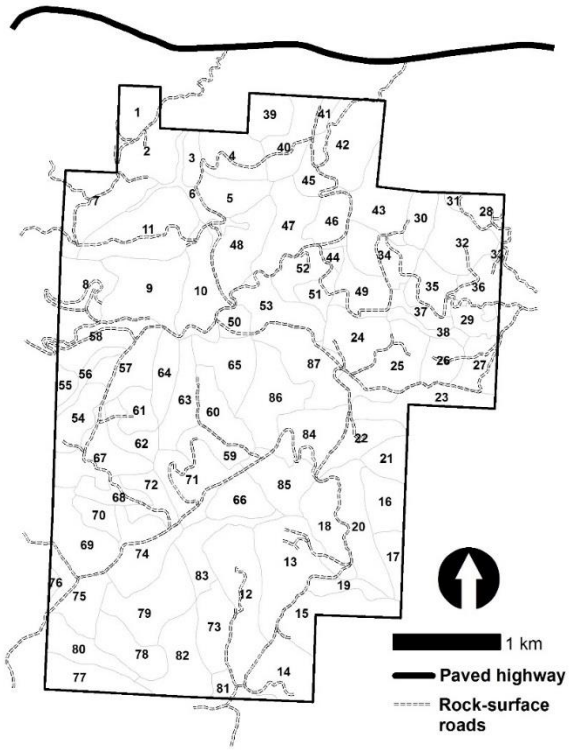


**Figure 3: Map of the Lincoln Tract**

schedules. The first method presented is an implementation of a greedy local search to repair invalid solutions with the aim of improving them. The second method is a stochastic repair method that does not take the value of a solution into account during repair. Both of these methods were used in conjunction with the genetic algorithm, a population based

[2] Submitted to ICAI 2015 WorldComp Conference Las Vegas

stochastic optimization method. Both methods produced good results, with several important differences in the quality of their solutions.

Section 1 of this paper introduces harvest scheduling and the genetic algorithm. Section 2 explains the repair methods that are being compared, section 3 describes the experimental method, and the results are presented in section 4.

## Section 1.1 Harvest Scheduling:

The goal in creating a harvest schedule is to minimize the output of an objective function that represents how good any arbitrary schedule is. This allows schedules to be compared with a numerical basis.

The Lincoln Tract is a parcel of timber-land in Washington State. Foresters break up parcels of land such as the Lincoln Tract into smaller units they call stands. Stands are usually separated by easily identified geographical features such as rivers, hills, a change in tree type, or previously cut roads.

The Lincoln Tract consists of 87 stands. Before any trees are cut, a plan must be constructed to harvest the timber. Experts who construct harvest schedules for timber must consider several important factors: how much timber will be harvested, ecological impact, and local laws. The optimization objectives for the Lincoln Tract were first described in *Forest Management and Planning* [1].

### *Harvest Schedule Objectives:*

1. 30 Year Harvest Cycle
2. 6 Harvest Periods spaced every 5 years.
3. Even Flow Harvest Target Value of: 13,950 MBF (million board feet)

In addition to these three harvest objectives, there are also several hard constraints that a schedule must follow:

1. Adjacent Cut Areas cannot exceed 120 Acres.
2. No stand can be cut before it is mature (30 year maturity threshold).
3. Stands 6, 38, and 58 cannot be harvested.

4. No more than 50% of the forest may be harvested.

The fitness function used in all of these experiments can be mathematically expressed this way:

$$Minimize \sum_{i=1}^{n} (H_i - T)^2$$

- i is the harvest period
- n is the number of harvest periods (6)
- $H_i$ is the total harvest in period $i$
- $T$ is the target harvest (13,950)

The function is squared to adjust for the effect of harvests that harvest too much and have a negative value and harvest schedules that harvest too little and have a positive value. All harvests that miss the target harvest value contribute to error, regardless of overcut or undercut.

## Section 1.2 Optimizing using A Modified Genetic Algorithm:

The combinatorial complexity of a scheduling problem such as this one is daunting. Added to that complexity is the fact that many of the possible schedules will violate one or more of the constraints outlined above. A problem of this complexity benefits from the use of an optimization technique.

Finding an optimal solution to this problem is NP-Hard. There are many general optimization algorithms that can be applied to this class of problem. Previously, both Genetic Algorithms and Particle Swarm Optimization have shown some promise [2].

Some variation on greedy repair has been used for harvest scheduling as early as 1995 [3]. Liu's algorithm was applied to a much simpler harvest scheduling problem with a different set of objectives, but the greedy repair combined with another greedy operator yielded good results. Later, Bäck performed a similar comparison of operators on the set

covering problem with constraints and again found that repair operators yielded stronger results than the penalty method for GAs [4].

These earlier problems showed promising results, but their problems were order of magnitudes simpler than the scheduling problem tackled here. Bettinger's work with Raindrop Optimization as applied to harvest scheduling has shown promise for repair operators as well [5].

The Genetic Algorithm (GA) is a stochastic, population based meta-heuristic used for optimization. The initial algorithm was developed by Holland in 1975 [3]. The GA is a general optimization algorithm that can be applied to many optimization scenarios and loosely mimics the process of natural selection. In this case, each schedule is analogous to the genome of an individual organism in a population. Better mates are chosen for mating, and their offspring produce stronger offspring in turn.

For our problem, each harvest schedule is represented as an array of length 87 with integers 1-6 representing harvest times and zero representing a no-cut scenario.

[1,4,0,2,3,6,0…..]

A problem arises when two chromosomes are crossed: they may produce unviable offspring. In nature, this individual would die, but in a computer simulation, this often represents a waste of good information. Repair and penalty functions are the two most popular methods for helping the genetic algorithm produce valid solutions.

In the repair method, small changes are made to solutions that are violating one or more of the constraints, to eliminate their constraint violation. Usually, some deeper knowledge of the problem must be known to produce good repair. The repair methods described later rely very little on intimate knowledge of the problem, and could therefore

be used on any problem where a) valid assignments can be identified independently and b) the impact of those changes can be measured.

Penalty functions put pressure on the algorithm to produce individuals that obey constraints. This is done by adding a penalty value to the fitness of any individual that violates constraints. We have taken the penalty function method as our baseline comparison because like the repair methods presented here, it can be implemented without much domain specific knowledge and it is a popular solution.

# Section 2.0 The Algorithms

## Section 2.1 Stochastic Repair:

The stochastic repair algorithm developed for these experiments is a fairly simple algorithm with two goals: 1. Eliminate bias during repair and 2. Bring the schedule back within the bounds of the hard constraints without creating any new violations. The first goal is achieved by identifying all of the stands currently in violation, then randomizing their fix order. The second goal is achieved by fixing the violation of each of these stands progressively until the entire schedule is back in compliance with the hard constraints. No consideration is given to whether these fixes improve the fitness of the solution beyond making them viable. Because there is an option which cannot violate any of the constraints (assigning a stand to a no-cut) both of the repair algorithms are guaranteed to stop.

### Stochastic Repair Algorithm:

While (Schedule Violates C1->Cx)

```
List<Stand> Violators = Stands in violation of any constraint.
Stand X = Violators(Random Stand);
List<TimePeriod> Legal = All Periods that Are Legal for X and no-cut;
TimePeriod Y = Legal.ChooseRandom;
StandX.TimePeriod = Y;
```

The first step of the repair is to identify the stands that are currently causing a violation. Because the algorithm never allows a stand that is too young to be cut, we only have to check for violations caused by stand-groups that exceed our limit for size. If stand A,B,C together violate our aggregate rule, but we remove A from the assignment, it is possible for B and C to then come into compliance with our rule. Therefore, we only fix a single stand at a time, we do not want to disrupt the schedule any more than necessary.

It is important to choose the stand to repair at random, not by its placement in the list of stands in violations. Choosing a stand at random prevents us from biasing our schedules arbitrarily by the order in which we have assigned our stands in a list. The structure of our list should not have arbitrary influence on our outcome. For the same reason, we do not want to bias the algorithm towards choosing a cut period over the no-cut option. Therefore, the no-cut option is always available as an assignment.

## Section 2.2 Greedy Stochastic Repair:

The first of these principals – avoiding order bias is preserved in the greedy stochastic repair, but by its nature the greedy algorithm is biased in its selection of assignments. The greedy repair algorithm used for these tests is a simple way to attempt to strategically minimize the outcome of a schedule while also satisfying the hard constraints of the problem. Certainly, the idea of greedy local search with stochastic elements is not new. Stochastic local search algorithms such as GSAT [7] have been studied since the 80s. This algorithm combines greedy local search with a stochastic ordering to help prevent order bias.

### Greedy Repair Algorithm:

```
While (Schedule Violates C1->Cx):
    List<Stand> Violators = Stands in violation of any constraint.
        Stand X = Violators(Random Stand);
    List<TimePeriod> Legal = All Periods that Are Legal for X and no-cut;
    TimePeriod G = From Legal Select Stand G Where G Minimizes
    Objective Function the Greatest;
    StandX.TimePeriod = Greedy;
```

The greedy repair algorithm shares many features with the stochastic version of the algorithm. A list of stands that violate any constraints is created, then one stand is randomly selected for repair. All of the legal cut period assignments for the randomly chosen violator are then aggregated. It is at this point that the greedy algorithm deviates from the stochastic. Each of the legal assignments for the stand are then evaluated and the assignment that minimizes the objective function the most is chosen as the assignment for that stand.

### Section 2.3 Proportional Penalty

The proportional penalty algorithm relies on the basic GA framework that all of the solutions share, except instead of repairing solutions that violate one of the constraints, it adds a penalty to the objective function. A penalty function that is too high can prevent a GA from exploring through invalid territory, but a penalty function that is too low will prevent a GA from converging on a legal solution [8].

## Section 3 Experimental Design:

After running some initial tests for tuning the Genetic Algorithm's parameters, the following parameter settings were found to yield good results for both the Greedy and Stochastic repair operators. Crossover was set to .9, mutation rate was set to .005, a maximum of 5000 generations was used, elitism, and tournament selection for crossover, with a tournament size of 3. Both algorithms performed better with low mutation rates.

Although the algorithm often converged well before generation 5000, some of the best results in the test sets did not reach convergence until beyond the 4000$^{th}$ generation. The extra generations added time to the experiment, but did not affect the results of the runs where convergence happened earlier. The algorithm also performed poorly with mutation rates over .01 and below .005. Although the algorithm seems sensitive to disruption from mutation, it could not explore enough after convergence to find good answers when the mutation rate was set too low.

The schedule representation was a list representing the 87 stands. Each stand could be assigned an integer 0-6. A 0 represents a no-cut and 1-6 represent the periods in 5 year increments. Each schedule was initialized with a random, but valid assignment. Since each schedule had valid age-requirements to begin with and mutation was also restricted to valid age requirements, the repair algorithms only had to deal with the second hard constraint, which is that no aggregation of cut stands can be over 120 acres.

A baseline performance measure was established through running the algorithm with the penalty scheme described earlier. Several penalty functions were tested, but the proportional scheme was chosen because it faired the best in our test runs. It was run 10 times with population sizes of 1000, 2000, 3000 and 4000 respectively.

Both repair algorithms were then run 10 times each with increasingly large population sizes. It was hypothesized that the greedy algorithm would perform better than the stochastic algorithm with smaller population sizes because it would be more able to find good answers with sub-par starting points. The algorithms were exercised multiple times because Genetic Algorithms and the repair algorithm employed in this experiment are both non-deterministic and it is possible to get very good and very bad answers with

the same technique. Running multiple times also yields valuable information about the reliability of results from the algorithms.

# Section 4 Results

## Section 4.1 Baseline

The baseline results were generally good, with a population size of 3000 yielding the best overall results, with the exception of a single outlier that can be seen in the figure below.

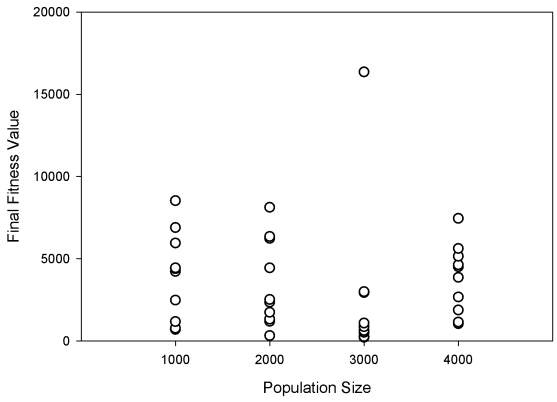


**Figure 4: Baseline Results for Lincoln Tract Problem**

The clusters of answers were fairly tight with a population size of 3000 again showing the tightest clustering along with the best results for our baseline experiment.

## Section 4.2 Greedy Repair

The results of Greedy Repair were far more consistent than the unbiased method. This is likely due to similar optimizations being achieved at early stages that push the algorithm towards a local optima instead of the global best. However, it also means that good, if not great, results can be achieved in fewer executions of the algorithm. The consistency, and good results with smaller populations, could outweigh the loss of best results under some circumstances. The difference between best results and worst results

with a population of 4000 is just over 1000, if a single outlier is omitted. This represents a very tight clustering.

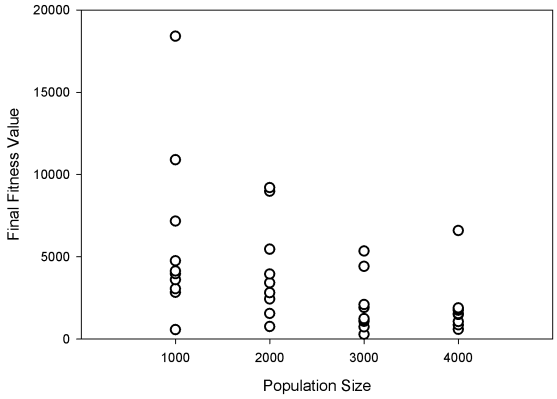


**Figure 5: Greedy Repair Results**

## Section 4.2 Stochastic Repair

The results from stochastic repair are promising. The algorithm was able to find very good solutions to the scheduling problem, with the best result being 66. With large population sizes good results were common, but the variation in the results tended to be large. Our best achieved results were with population size of 4000 and the fitness values between best and worst varied by more than 5000, showing that the consistency of the stochastic algorithm was not close to the consistency of the greedy algorithm. However, the algorithm produced the best result recorded during these experiments at a fitness of 66.02, which is very good. The algorithm trades off the chance at great results for consistency between runs.

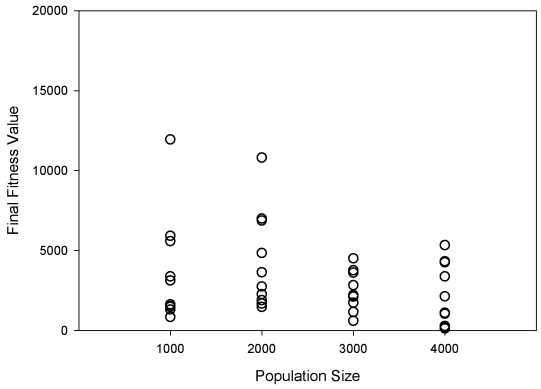


**Figure 6: Stochastic Repair Results**

## Section 5 Conclusion:

Both repair algorithms tested yielded good results on the problem set with sufficiently large populations, but showed different characteristics with respect to consistency and best results observed. Their performance strengths were different, with stochastic yielding better "best" results while the greedy algorithm was more consistent. This pattern holds when compared to the baseline penalty scheme as well.

**Table 1. Summarized Experimental Results**

| Metric | Baseline | Greedy | Stochastic |
|---|---|---|---|
| Best | 223.811 | 586.49 | 132.659 |
| Worst | 16354.658 | 6589.19 | 5341.923 |
| Average | 2645 | 1910.3 | 2226.094 |
| Standard Deviation | 4671.124 | 1612.06 | 1857.502 |

The penalty method does not yield results quite as good as the stochastic method, nor is it as consistent as the greedy method. The simple stochastic repair method is preferable on many grounds because it doesn't add much computation time and yields good results. However, if one is constrained by time and cannot run many trials, the greedy

method may be better because it consistently gives good results. In either scenario, a repair method seems to be a better choice over the penalty method for this problem.

## REFERENCES

[1] P. Bettinger, K. Boston, J. Siry, and D. Grebner. Forest Management and Planning, 1st Edition. Elsevier, 2008.

[2] K. Deb. "An efficient constraint handling method for genetic algorithms. Comput. Methods Appl. Mech Engrg. 186 (2000) 311-338 Elsevier.

[3] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[4] G. Liu, "Evolution Programs, Simulated Annealing and Hill Climbing Applied to Harvest Scheduling Problems", Thesis, University of British Columbia, 1995.

[5] W.D. Potter, E. Drucker, P. Bettinger, F. Maier, M. Martin, D. Luper, M. Watkinson,

[6] G. Handy, and C. Hayes, "Diagnosis Configuration, Planning and PathFinding: Experiments in Nature-Inspried Optimization," in Natural Intelligence for Scheduling, Planning, and Packing Problems, edited by R. Chiong, Springer-Verlag, Studies in Computational Intelligence (SCI) series, 2009.

[7] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp. 548–553, IEEE, June 1994.

[8] B. Selman, H. Levesque, and D. Mitchell (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446.

# CHAPTER 4

# CONCLUSIONS

In this thesis, an exploration of methods to solve some of the basic issues with stochastic optimization were presented. In Chapter 2, a simple algorithm for optimization that was inspired by the act of fishing from a boat was presented. In Chapter 3, two repair algorithms for a harvest scheduling problem were presented with comparisons to a penalty method, which served as a baseline for the evaluation of the repair algorithms.

The development of the EPA algorithm was mostly an exercise in creation. The algorithm evolved as it was implemented and tested until the simple framework yielded good results. In the end it was not as powerful or as efficient as the Genetic Algorithm on large problems, but it fared well across all problems. It is likely that the EPA could undergo further refinement and improvement to make it a more efficient searching algorithm.

The repair methods presented in Chapter 3 both present improvements over a penalty-based method, depending on whether the optimization professional is emphasizing better results over several runs or more consistent good results. Further research into repair methods, especially those that use a predictable repair scheme without introducing negative bias into the problem would be very useful. Although repair is considered by some to be counter to the activity of the Genetic Algorithm framework, it has proven to be a useful tool for getting good answers within the valid search space for scheduling problems.

## REFERENCES


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

[2] Kennedy, J. & Eberhart, R.C. 1995. Particle swarm optimization. Proc. IEEE International Conference on Neural Networks, Picataway NJ: IEEE Service Center, 1995, 1942-1948.

[3] Mathews, G. B. (25 June 1896). "On the partition of numbers". Proceedings of the London Mathematical Society 28: 486–490.

[4] Dantzig, G. 1963. *Linear Programming and Extensions*. Princeton University Press, West Sussex.

[5] F. Glover. "Tabu Search—part I." ORSA Journal On Computing , Issue 3, 190-206, 1997.

[6] S. Boettcher & A. G. Percus. " Extremal Optimization : Methods derived from co-evolution." Modeling and Simulation Design. AK Peters Ltd. 1999.

[7] S. Kirkpatrick, S.&Vecchi, M.C.. "Optimization by simmulated annealing." Science, 220(4598), 671-680, 1983.

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

[9] Kenedy, J. &Eberhart, R.C. 1995. Particle swarm optimization. *Proc. IEEE International Conference on Neural Networks,* Picataway NJ: IEEE Service Center, 1995, 1942-1948.

[10] D. T. Pham, E. Ghanbarzadeh, A. Koc, E. Otri, S. Rahim, S. &Zaidi, M. 2009. The Bees Algorithm – A Novel Tool for Complex Optimisation Problems.*Innovative Production Machines and Systems,2009.*
*DOI=http://conference.iproms.org/the_bees_algorithm_a_novel_tool_for_complex_optimisation_problems.*

[11] Dorigo, M., & Di Caro, G. 1999. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (Vol. 2). IEEE.

[12] Potter, W. Drucker, E. Bettinger, P. Maier, F. Luper, D. Martin, M. Watkinson, M. Handuy, G. &Hayes, C. 2009. Diagnosis, Configuration, Planning, and Pathfinding: Experiments in Nature-Inspired Optimization.*Natural Intelligence for Scheduling, Planning and Packing Problems,* R. Chiong, ed., Springer-Verlag.

[13] FL Chang and W. Potter. A genetic algorithm approach to solving the battlefield communication network configuration problem. In: Yfantis, EA (ed) Intell Sys. Third

GoldenWest Intern Conf (Theory and Decision Library D, Vol 15). Kluwer,Dordrecht. 1995.

[14] P. Bettinger, K. Boston, J. Siry, and D. Grebner. Forest Management and Planning, 1st Edition. Elsevier, 2008.

[15] G. Liu, "Evolution Programs, Simulated Annealing and Hill Climbing Applied to Harvest Scheduling Problems", Thesis, University of British Columbia, 1995.

[16] T. Bäck, M. Schütz, S. Khuri, "A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem," Artificial Evolution, Lecture Notes in Computer Science Voume 1063, 1996, pp. 320-332, 1996.

[17] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," European Journal of Operational Research, vol. 94, no. 2, pp. 392–404, 1996.

[18] B. Selman, H. Levesque, and D. Mitchell (1992). A new method for solving hard satisfiability problems. In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), pages 440–446.

[19] K. Deb. "An efficient constraint handling method for genetic algorithms. Comput. Methods Appl. Mech Engrg. 186 (2000) 311-338 Elsevier.

[20] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp. 548–553, IEEE, June 1994.