A FAMILY OF GENETIC ALGORITHM PACKAGES ON A WORKSTATION FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS*

Roger L. Wainwright
The University of Tulsa

Keywords: Genetic Algorithms, Combinatorial Optimization

Abstract

LibGA, HYPERGEN and GATutor represent a trio of in-house developed genetic algorithm software packages for conducting research in combinatorial optimization problems. For a variety of reasons, these packages were developed using the C programming language on a Unix platform, specifically a Sun workstation. We used these packages successfully in developing sequential and parallel genetic algorithms for solving numerous combinatorial optimization problems. Several of these combinatorial optimization research projects and their results are described.

Introduction

In theory, the algorithms in the class of NP-complete problems are considered computationaly equivalent. NP-complete problems have no known deterministic polynomial time algorithms; that is, there is no known solution except to try all combinations. These problems are sometimes referred to as combinatorial optimization problems. It is currently impossible to optimally solve any of these problems, except for trivial cases. In general, finding a solution requires an organized search through the problem space. An unguided search is extremely inefficient. Consequently, researchers have focused on approximation techniques which provide efficient, near optimal solutions. Some of the techniques which are applicable to NP-complete problems include heuristic techniques, tabu searching, simulated annealing, neural networks, and genetic algorithms (GA). In this paper I will concentrate on the genetic algorithm technique implemented on a workstation for solving combinatorial optimization problems.

* Research was supported by OCAST Grant AR2-004 and Sun Microsystems Inc.

Genetic Algorithms

There are several ways that genetic algorithms differ from traditional algorithms. Genetic algorithms are based on the principles of natural genetics and survival of the fittest. Genetic algorithms search through a solution space by emulating biological selection and reproduction. The GA works from a population of strings instead of a single point. The genetic algorithm works with a coding of the parameter rather than the actual parameter. The genetic algorithm uses probabilistic transition rules, not deterministic rules, and the applications of the genetic operators causes information from the previous generation to be carried over to the next. In addition, genetic algorithms produce "close" to optimal results in a "reasonable" amount of time. Furthermore, genetic algorithms are fairly simple to develop and they are suitable for parallel processing.

The genetic algorithm is a robust search and optimization technique using probabilistic rules to evolve a population from one generation to the next. The transition rules going from one generation to the next are called genetic recombination operators. These include Reproduction (of the more "fit" chromosomes), Crossover, where portions of two chromosomes are exchanged in some manner, and Mutation. Crossover combines the "fittest" chromosomes and passes superior genes to the next generation thus providing new points in the solution space. Mutation is performed infrequently. A new individual (point in the solution space) is created by altering some of the bits of an individual. Mutation ensures the entire state space will eventually be searched (given enough time), and can lead the population out of a local minima. Genetic algorithms retain information from one generation to the next. This information is used to prune the search space and generate plausible solutions within the specified constraints [2,14].

The genetic algorithm creates an initial population of feasible solutions, and then recombines them in a such way to guide the search to only the most promising areas of the

state space. In a *generational* GA the offspring are saved in a separate pool until the pool size is reached. Then the children's pool replaces the parent's pool for the next generation. In a *steady-state* GA the offspring and parents occupy the same pool. Each time an offspring is generated it is placed into the pool, and the weakest chromosome is "dropped off" the pool. These two cases represent two extremes in pool management for genetic algorithms. Several researchers have investigated the benefits of solving combinatorial optimization problems using genetic algorithms [3,7,8,9,13,16,22,23]. It is assumed the reader is familiar with the fundamentals of genetic algorithms.

Genetic algorithm packages for a single processor have been available for several years. A steady-state GA such as GENITOR [21] and a generational GA such as GENESIS [11] are two example packages that have been available for several years. The research reported here made use of LibGA [6], a GA package developed in house. LibGA offers the best of GENESIS and GENITOR including the ability to use a steady-state or a generational approach or a combination of both. Davis, Goldberg and Rawling provide an excellent in depth study of genetic algorithms [7,8,10,16].

Even though the fundamental concepts of genetic algorithms are fairly simple and straightforward, there are numerous implementation variations and options to incorporate into a genetic algorithm. For example, there are numerous selection techniques for determining chromosomes for crossover. There are many ways to parametrize a model and encode it into a finite length chromosome. There are several techniques for introducing mutation to a chromosome. There are also dozens of possible crossover operators that have been developed in recent years depending on the problem type and chromosome encoding scheme.

Genetic Algorithm Packages

GENESIS [11], written by John Grefenstette, was the first widely available genetic algorithm package. Since that time, a variety of genetic algorithm packages have been developed. Most of these use the generational model. However, a few use the steady-state model introduced with GENITOR [21] in 1988. For a current list of GA packages, the reader is referred to the GA Software Survey [17]. The survey contains more detailed information concerning what is available and how to obtain GA packages. Notice most of these packages were developed for the workstation environment and not for PC's. Genetic algorithms require a tremendous amount of CPU time for moderate to large sized problems, up to several days, perhaps. Several of the most popular GA packages are summarized below.

GAucsd [17] is a GENESIS-based genetic algorithm. It offers several bug fixes and an improved user interface. In addition, it can distribute the GA over a network of machines. Its most impressive feature is Dynamic Parameter Encoding (DPE). DPE is a technique which is used in continuous search spaces. It redefines the encoding used for the chromosome, adapting the 'granularity' to the convergence rate. GAucsd was designed primarily for use on bit string chromosomes, which does not work well with order-based problems. While GAucsd can encode the integers using grey-code bit strings, it does not ensure that order is preserved. As a result, this can produce chromosomes with invalid allele orderings or alleles with meaningless bit patterns. GAucsd uses a variation of the roulette wheel for selection, a bit inverting mutation, and a two-point crossover operator [6].

GENEsYs [17] is another popular generational genetic algorithm. It is based on GENESIS and includes several different selection schemes such as linear ranking, Boltzmann, (mu, lambda)-selection. The package includes uniform and n-point crossover as well as discrete and intermediate recombination. Mutation can be self-adaptive. Unfortunately, the user does not have the ability to test a steady-state model with these options.

GENITOR [21] is a steady-state GA which uses a rank-based, biased selection and weakest chromosome replacement. It has the ability to work with integer and floating point types as well as bit strings. GENITOR includes an example GA for the traveling salesman problem with order, PMX and edge-based crossover operators. It can also perform subpopulation modeling. We have used GENITOR successfully in the past for order-based problems [5]. However, steady-state genetic algorithms can converge prematurely. They require large pool sizes and many trials to ensure the best solution is found [6].

LibGA

The LibGA software package [6] was developed primarily because of the noticeable deficiencies of existing GA packages. Our previous research was limited to the steady-state GA used in GENITOR augmented with custom order-based crossover and mutation functions. The problems that steady-state GAs have with premature convergence led to the more commonly used generational genetic algorithm. We decided that modifying GENITOR to include a generational model was too great a task. Consequently, we elected to develop our own workstation based genetic algorithm package to meet the needs of our current research.

LibGA is a collection of routines written in the C programming language. It can run on a variety of PC's and workstations. However, since everything in LibGA is in double precision and since genetic algorithms are inherent-

ly CPU bound, we elected to implement LibGA in a workstation environment using Unix. We found that executing LibGA on a PC was extremely slow. LibGA provides a user-friendly workbench for order-based genetic algorithm research. It includes a rich set of genetic operators for selection, crossover and mutation. An important feature of LibGA is the ability to implement both a generational and steady-state genetic algorithms using the genetic operators. This allows researchers the ability to compare between the two approaches. Other features of LibGA include a generation gap, elitism, and the ability to implement a dynamic generation gap. Other routines are provided for initialization, reading a configuration file and generating various statical reports. Thus far, LibGA has been requested and sent to over a dozen locations in the United States and Europe.

LibGA includes operators for selection, replacement, crossover, and mutation. Selection and replacement can be augmented with elitism. This ensures that the best member of a population survives into the next generation. The selection operators included in LibGA are: *uniform-random*, *roulette*, *min-roulette*, and *rank-biased*. Uniform-random selection picks a member of the pool at random, completely ignoring fitness or other factors. Thus each chromosome in the pool is equally likely to be selected.

The replacement operators in LibGA are: *append*, *by-rank*, *first-weaker*, and *weakest*. The append replacement operator appends new chromosomes to an existing pool. This operator is used in the classical generational GA to place offspring in the new pool. The by-rank operator is the same as the one used in GENITOR. The pool is ranked by sorting the fitness values. If the chromosome has a "high" fitness, it will be placed in the pool, displacing "weaker" chromosomes. If its fitness is worse than the weakest member of the pool, it dies and is not placed in the pool. Note the weakest and first-weaker operators are somewhere between the append and by-rank operators.

LibGA's crossover operators include *simple*, *order1*, *order2*, *position*, *cycle*, *PMX*, and *asexual*. Simple crossover is used for traditional bit string encodings of the chromosome. In this case a random crossover point is selected which divides each parent chromosome into two parts. Alternate parts are contributed by each parent to generate two offspring. This is also known as single point crossover. This crossover operator does not work for order-based problems, however, since order is not preserved. The other crossover operators preserve order information. Order1, order2, position, cycle, and PMX operators are described in Starkweather *et al.* [20]. The asexual operator is a simple swap of two randomly selected genes, which also is suitable for order-based problems. LibGA currently offers the following mutation operators: *simple-invert*, *simple-random*, and *swap*.

HYPERGEN

HYPERGEN [12] was developed as a research tool for investigating parallel genetic algorithms applied to combinatorial optimization problems. HYPERGEN is a distributed genetic algorithm for a hypercube. The key concept to HYPERGEN is that it distributes the initial population evenly among the processors. Each processor (island) executes a sequential GA on its subpopulation performing crossover and mutation. HYPERGEN uses a steady-state reproduction scheme where only a few members of the population are removed at each iteration. The population pool size remains constant in each processor. After a prescribed number of reproductions, (called the *migration interval*) the "fittest" chromosomes in each processor are exchanged among other processors introducing new genetic material into each island. The amount of genetic material to exchange is called the *migration rate*. This process continues until the entire population stabilizes [12].

HYPERGEN was designed as a modular collection of routines for generating the initial population, evaluation function, selection, reproduction, mutation, migration interval, migration rate and summary statistics. A Key feature in HYPERGEN is that the sequential GA on each processor and the periodic migration of genetic material between processors is performed automatically for the user. One of the design requirements for HYPERGEN is that the user can use the package without concerning himself with the hypercube topology, message passing, or other details of parallel processing.

To drive the HYPERGEN system, the user must provide the evaluation function, which defines the problem to be solved. The user also specifies the following parameters (most of which have defaults, if not specified): (a) input dataset describing the problem, (b) output dataset (optional), (c) seed the initial population (optional), (d) dimension of the hypercube, (e) random number seed, (f) population size (per node), (g) bias parameter for selection, (h) the migration interval, (i) the migration rate, (j) how often to collect and report statistics, (k) maximum migrations allowed, (l) select crossover function, and (m) select mutation function [12].

HYPERGEN allows the user to input a dataset describing the problem. For example, a sequence of coordinates describing locations of cities for the Traveling Salesman Problem. Results can be sent to a file or default to the screen. The user also has the option to seed the initial population or allow for a random initial population to be generated. The migration interval, default of 20, specifies how many local reproductions are to occur in each node between migrations. The migration rate, default 20%, is the percent of the population in each node that is exchanged with another node. The bias parameter is used to

indicate how much more likely is the best chromosome to be selected over the median chromosome. We developed a feature that allows for an adaptive migration interval and for an adaptive migration rate. This means the migration rate and interval are monitored and altered depending on the conditions of the population. For example different interval and rates may work better as the population matures. Genetic material is exchanged during each successive migration along the dimensions of the hypercube.

HYPERGEN was implemented for order-based genetic algorithms. An order-based GA is where all chromosomes are a permutation of a list. Order-based GAs can be applied to a wide range of combinatorial optimization problems such as the following classic problems: Bin Packing, TSP, Package Placement, Job Scheduling, Vehicle Routing, Network Routing, and various layout problems, etc. The HYPERGEN system supplied crossover functions for order-based GAs include Edge Recombination, Order Crossover #1, Order Crossover #2, PMX, Cycle Crossover, Position Crossover. The Mutation functions available in HYPERGEN include swapping two elements, moving one element to another location, and reordering a sublist. HYPERGEN was used successfully to find new "best" tours on three "standard" TSP problems, and out performed our parallel simulated annealing algorithm on various Package Placement Problems [12].

Intel Corporation has developed an iPSC/2 hypercube and iPSC/860 hypercube simulator that runs on a Unix based workstation. We have implemented HYPERGEN on the Intel simulator. Thus, most of our initial software development using HYPERGEN to solve various problems has been developed on a Sun workstation. The CPU demands and an operating system that allows for pipes and sockets to simulate a hypercube makes it necessary to use a powerful workstation. The simulator has been extremely valuable. Only after considerable testing do we port the code to a hypercube multiprocessor.

GATutor

GATutor [15] is a graphical tutorial system for genetic algorithms. The X Window/Motif system provides powerful tools for the development of the user interface with a familiar feel and look. We implemented the Set Covering Problem (SCP) and the Traveling Salesman Problem (TSP) as two example GA problems in the tutorial. The TSP problem uses an order-based chromosome representation (permutation of n objects), while the SCP uses bit strings. On the screen layout of the tutorial, the user has numerous buttons to select the GA parameters. These include (a) population size, (b) type of initial population (random or from a file), (c) selection bias, (d) maximum number of generations (trials), (e) generation gap, (f) selection mode (roulette, etc.), (g) replacement method, (h) mode (steady-state or generational), (i) mutation method, (j) mutation rate, (k) selection of the crossover operation from a choice of several possibilities, (l), elitism, etc. The user has the ability to do a step by step execution or to do a continuous run. We developed the screen layout to provide a visual representation of the chromosomes in the population with the ability to scroll. This gives the user the option of varying one or more GA parameters to visually see the effect on the algorithm. An important feature of this tutorial is the set of help screens that explain (with examples) all of the options for each of the GA parameters.

GATutor was designed to assist students in learning the foundations and principles of genetic algorithms and as a research tool in developing genetic algorithms. Specifically, GATutor was: (1) Designed to be used by anyone after a typical introduction lecture on genetic algorithms. The lecture may not necessarily include any instruction on GATutor at all. (2) Designed for a general audience, not just computer science students. Thus, no programming is required and very few computer skills are required by the user. (3) Portable and easy to install. (4) Modular enough to easily be able to add new features and options to the package by different people over several years. (5) Comprehensive, with a full set of help menus to explain (with examples) all of the numerous GA terms, definitions, operators and parameters, and (6) A self contained package where students can visually see the effect of executing a genetic algorithm.

Based on the design criteria, we elected to implement GATutor using the X Window/Motif system with the C programming language on a Unix based platform. The speed required for a rapid feedback tutorial, and the enormous CPU time required to run a moderate to large GA led us to develop the package on a Sun Sparc 10 Workstation instead of a PC.

Combinatorial Optimization Problems

Added to LibGA and HYPERGEN, GATutor completes our trio of in-house developed GA software for doing research in combinatorial optimization problems. We have used these packages successfully in developing GAs for solving numerous combinatorial optimization problems. Several of these research problems are described below.

1) "A Genetic Algorithm for Packing in Three Dimensions" [5]. Recent research in Bin Packing has almost exclusively been in two dimensions. In this research, we extend the classic Bin Packing problem to three dimensions. We investigated the solutions for the three dimensional packing problem using first fit and next fit packing strategies with and without genetic algorithms. Five data sets were used to test our algorithms, both random and contrived. They range from 50 to 500 packages. We also

studied several existing crossover functions for the genetic algorithm: PMX, Cycle, and Order2. A new crossover function, Rand1, was developed. The genetic algorithm was tested using a randomly generated initial population pool, and a seeded initial pool. The seeded pool was generated from a package ordering produced by rotating and sorting the packages by decreasing height. Results showed the seeded genetic algorithm using Next Fit and PMX produced the best overall results for the data sets tested. The seeded genetic algorithm using Next Fit and Order2 provided the best results considering both rapid execution time and packing efficiently. We found genetic algorithms to be an excellent technique for yielding good solutions for the three dimensional bin packing problem.

2) "Multiple Vehicle Routing with Time Windows using Genetic Algorithms" [2]. In this research we investigated genetic algorithms as a heuristic technique for obtaining optimal or near optimal solutions to the single and multiple vehicle routing problem with time windows and capacity constraints (VRPTW). The traditional crossover operators for order-based genetic algorithms are not well suited for optimization problems with multiple constraints. We developed and presented two new crossover operators, Merge Cross #1 (MX1) and Merge Cross #2 (MX2). These new operators actually produce a family of new crossover operators. The Merge Cross operators are based upon the notion of a global precedence among genes independent of any chromosome. We compared the Merge Cross family of operators with several well known crossover operators: PMX, Cycle and Edge Recombination. We tested our MX Crossover operators on three randomly generated models of 15, 30, and 75 customers. A fourth model of 99 customers was supplied to us by a local retail distribution company in Tulsa, Hale-Halsell Company. In each instance one of our new MX operators was the best performer. We showed that all of the MX operators are excellent crossover operators for solving problems where there exists a global precedence relationship among the genes. In the 99 customer problem, the local retail distribution company used 39 vehicles, while our algorithms yielded a 30 vehicle solution.

3) "Manipulating Subpopulations of Feasible and Infeasible Solutions in Genetic Algorithms" [18]. This research explores the partitioning of the population pool in genetic algorithms into separate subpopulations of feasible and infeasible solutions, and the interaction on a regular basis of crossover operations among and within the subpopulations. The Set Covering Problem was chosen as a representative optimization problem to apply our subpopulation strategies. We designed nine algorithms for manipulating the two population pools and compared this against the traditional GA. The traditional GA uses a single population pool where infeasible solutions are generally considered infrequently or ignored. All of our algorithms significantly and consistently outperformed the traditional GA in all of the test problems illustrating the importance of infeasible solutions as a source of good genetic material. Furthermore, results show that the random select consistently outperformed the bias select from the infeasible pool suggesting all infeasible solutions should be considered equal.

4) "Manipulating Subpopulations in Genetic Algorithms for Solving the k-way Graph Partitioning Problem" [19]. This research also explores the partitioning of the population pool in genetic algorithms (GA) into separate subpopulations of feasible and infeasible solutions, and the interaction of crossover operations among and within the subpopulations. The Graph Bisection Problem (GBP) and the k-way Graph Partitioning Problems (k-GPP) were chosen as representative optimization problems to apply our subpopulation strategies. We designed several algorithms for manipulating the two population pools and compared this against the traditional GA. The traditional GA uses a single population pool where infeasible solutions are generally considered infrequently or ignored. We also developed several new crossover operators to be used while manipulating feasible and infeasible solutions.

We define an *X*-crossover as one that involves two chromosomes from the feasible pool. A *Y*-crossover involves one chromosome from the feasible pool and one from the infeasible pool. A *Z*-crossover involves two chromosomes from the infeasible pool. In all cases the resulting feasible solutions are placed into the feasible pool and the resulting infeasible solutions are placed into the infeasible pool. We designed a class of algorithms for manipulating the feasible and infeasible population pools based on the *X,Y,* and *Z* crossover options. We also developed a class of oscillation algorithms that varies *x* and *y* percentages with each new trial. We compared the difference in executing a problem using a bias selection from the infeasible pool, and a random selection. We also developed several new crossover operators specifically for this problem using the infeasible and feasible pools: *Modified Uniform*, *Modified Asexual*, and *Asexual Uniform* crossover operators.

We generated several datasets to test our GA implementation of the k-GPP problem. The performance of the traditional GA was hindered by not taking advantage of an infeasible pool. All of our algorithms significantly and consistently outperformed the traditional GA in all of the test problems. This illustrates again the importance of infeasible solutions as a source of good genetic material. Furthermore, results show that two of our new crossover operators consistently out performed the others.

5) "Detecting Multiple Outliers in Multidimensional Data Using Genetic Algorithms" [4]. Detection and treatment of outliers tremendously improves the results of normal statistical analysis. However, the detection of large numbers of outliers in a large dataset is a combinatorial

optimization problem. We used a genetic algorithm to detect outliers in three sample datasets with great success. The GA was a vast improvement over conventional outlier detection algorithms. In this research we described in great detail how GAs are applied to a given dataset, how the chromosome is constructed, and which crossover functions proved better than others. To our knowledge this work represents to first time GAs have been applied to outlier detection.

6) "The Design of a Multipoint Line Topology for a Communication Network Using Genetic Algorithms" [1]. In this research we investigated genetic algorithms as a heuristic technique for obtaining a near optimal solution for the multipoint line topology (MLT) problem. The MLT problem involves determining minimum cost links for a collection of terminal sites that communicate with a central site in a communication network. The problem is to determine a minimum cost tree, rooted at the central site and spanning all terminal sites. Each subtree rooted at the central site corresponds to a multipoint line interconnecting the central site with a subset of the terminal sites. The constraint is that the aggregate capacity requirement of the terminals in each multipoint line cannot exceed the capacity of the corresponding multipoint line. This problem is more generally known as the Constrained Minimum Spanning Tree (CMST) problem. The problem is NP-complete. The Esau-Williams algorithm is widely used to obtain a near optimal minimum cost spanning tree that satisfies the constraint.

Comparison between the Esau-Williams algorithm and different GA representations and crossover strategies were made in this research. The terminal locations were randomly generated and Euclidian distance was used as the cost. A randomly generated capacity was associated with each terminal. The best performing GA representation was an adaptation of the order-based Davis encoding method. Our genetic algorithms outperformed the Esau-Williams algorithm in all cases tested.

7) "Near-Optimal Triangulation of a Point Set using Genetic Algorithm" [24]. This research explores the triangulation of a point set using genetic algorithms. We implemented several GAs including a parallel genetic algorithm (HYPERGEN) for solving the triangulation problem. We developed a crossover operator specifically for the triangulation problem. Various data structures needed to implement the crossover operator were developed. Since our crossover operator is also the mutation operator, we investigated the effect of our genetic algorithms with and without mutation. We compared our genetic algorithms against the best known heuristic algorithm and a simulated annealing implementation for the triangulation problem. We tested all of the algorithms using randomly generated datasets of various sizes and some contrived datasets.

Our results indicate the GA using mutation performs better, in general, than our GA without using mutation. This signifies that mutation (even though it is the same as the crossover operator) is significant, especially as the state space size increases. Considering overall performance, the ranking of the algorithms we tested are: parallel GA (best), greedy (second), and generational GA with mutation (third). The simulated annealing algorithm was the worst algorithm. We attribute the superior performance of the parallel GA over the sequential GAs to the separately maintained population pools allowing for the independent breeding of chromosomes in separate islands. The parallel GA allows for an excellent mix of the chromosomes insuring a more rigorous search through the state space.

The workstation based packages, LibGA, HYPERGEN and GATutor have proved invaluable to us in our development and study of genetic algorithms for solving numerous combinatorial optimization problems.

LibGA, HYPERGEN and GATutor are available at no cost by writing or sending email to the author. The author's address is Department of Mathematical and Computer Sciences, The University of Tulsa, 600 South College Avenue, Tulsa, Oklahoma 74104-3189,
rogerw@penguin.mcs.utulsa.edu

Acknowledgements

This research has been supported by OCAST Grant AR2-004. The authors also wish to acknowledge the support of Sun Microsystems Inc.

References

[1] Abuali, F.N., Schoenefeld, D.A. and Wainwright, R.L. "The Design of a Multipoint Line Topology for a Communication Network Using Genetic Algorithms", *Seventh Oklahoma Conference on Artificial Intelligence*, November, 1993.

[2] Blanton, J.L. and Wainwright, R.L. "Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms", *Proceedings of the Fifth International Conference on Genetic Algorithms* (ICGA-93), Stephanie Forrest, Editor, Morgan Kaufmann Publisher, 1993, pp. 452-459.

[3] Brown, D.E., Huntley, C.L. and Spillane, A.R. "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[4] Crawford, K.D., Wainwright, R.L, and Vasicek, D.J., "Detecting Multiple Outliers in Multidimensional Data Using Genetic Algorithms", in review.

[5] Corcoran, A.L. and Wainwright, R.L. "A Genetic Algorithm for Packing in Three Dimensions", *Proceedings of the 1992 ACM Symposium on Applied Computing*, March 1-3, 1992, pp. 1021-1030, ACM Press.

[6] Corcoran, A.L. and Wainwright, R.L., "LibGA: A User-Friendly Workbench for Order-Based Genetic Algorithm Research", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, February, 14-16, 1993, pp. 111-117, ACM Press.

[7] Davis, L. ed., *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publisher, 1987.

[8] Davis, L. ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[9] De Jong, K.A. and Spears, W.M., "Using Genetic Algorithms to Solve NP-Complete Problems", *Proceedings of the Third International Conference on Genetic Algorithms*, June, 1989, pp. 124-132.

[10] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[11] Grefenstette, J., GENESIS, Navy Center for Applied Research in Artificial Intelligence, Navy research Lab., Wash. D.C. 20375-5000.

[12] Knight, L.R. and Wainwright, R.L. "HYPERGEN - A Distributed Genetic Algorithm on a Hypercube", *Proceedings of the 1992 IEEE Scalable High Performance Computing Conference*, Williamsburg, VA., April 26-29, 1992, pp. 232-235, IEEE Press.

[13] Liepins, G.E. and Vose, M.D. "Deceptiveness and Genetic Algorithm Dynamics, *Foundations of Genetic Algorithms*, G. Rawling, ed., Morgan Kaufmann Publishers, 1991.

[14] Mutalik, P.M., Knight, L.R., Blanton, J.L. and Wainwright, R.L. "Solving Combinatorial Optimization Problems Using Parallel Simulated Annealing and Parallel Genetic Algorithms", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, March 1-3, 1992. pp. 1031-1038, ACM Press.

[15] Prince, C., Wainwright, R.L., Schoenefeld, D.A. and Tull, T., "GATutor: A Graphical Tutorial System for Genetic Algorithms", in review.

[16] Rawling, G. ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, 1991.

[17] Schraudolph, Nicol N., Genetic algorithm software survey. Available by anonymous ftp from cs.ucsd.edu as /pub/GAucsd/GAsorf.txt, August, 1992.

[18] Sekharan, D. Ansa and Wainwright, R.L., "Manipulating Subpopulations of Feasible and Infeasible Solutions in Genetic Algorithms", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, February, 14-16, 1993, pp. 118-125, ACM Press.

[19] Sekharan, D. Ansa and Wainwright, R.L., "Manipulating Subpopulations in Genetic Algorithms for Solving the k-way Graph Partitioning Problem", *Seventh Oklahoma Conference on Artificial Intelligence*, November, 1993.

[20] Starkweather, T., McDaniel, S., Mathias, K. Whitley, D. and Whitley, C. "A Comparison of Genetic Sequencing Operators", *Proceedings of the Fourth International Conference on Genetic Algorithms*, June, 1991, pp. 69-76.

[21] Whitney, D. and Kauth, J., GENITOR: A Different Genetic Algorithm, *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, Co., 1988, pp. 118-130.

[22] Whitney, D., Starkweather, T. and Fuquat, D., "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator", *Proceedings of the Third International Conference on Genetic Algorithms*, June, 1989.

[23] Whitley, D. and Starkweather, T. "GENITOR II: A Distributed Genetic Algorithm, *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1990) 189-214.

[24] Wu, Yu and Wainwright, R.L., "Near-Optimal Triangulation of a Point Set using Genetic Algorithm", *Seventh Oklahoma Conference on Artificial Intelligence*, November, 1993.