# HYBRID SEARCH ALGORITHMS

Roy P. Pargas, Jennifer Ludwick, and Steven Spoon
Department of Computer Science
408 Edwards Hall
Clemson University
Clemson, SC 29634-1906
email: pargas@cs.clemson.edu

Dynamical systems, optimization, parallel processing

**ABSTRACT**

When employing search heuristics such as genetic algorithms, obtaining the optimal solution is never guaranteed. The challenge to the algorithm designer is to be creative in developing the algorithm components: selection, recombination, mutation. This paper presents an additional tool available to the genetic algorithm designer: combining genetic algorithms with traditional exhaustive search algorithms to form *hybrid* search algorithms. Hybrid algorithms allow the algorithms to cooperate with one another, providing clues which ideally accelerate the other's ability to find the optimal solution. Three search techniques are described in this paper. The problem studied is the Steiner tree problem, a variation of the minimum spanning tree problem A hybrid algorithm employing a technique called *extraction* is shown to produce optimal solutions significantly faster than a conventional exhaustive search algorithm for very large problems. A second technique, called *recycling*, enables a genetic algorithm to find optimal solutions consistently for difficult Steiner tree problems. Finally, a technique called *seeding* is described. The paper concludes with a description of further work planned for the general approach of hybrid search algorithms.

## I. INTRODUCTION

The problem addressed in this paper is the Steiner tree problem, a variation of the minimum spanning tree problem. Given a connected graph $G$, any tree consisting solely of edges of $G$ and including all vertices in $G$ is called a *spanning* tree. If the edges of $G$ are weighted, a spanning tree with least total weight is called a *minimum spanning* tree. It is interesting to note that if one is allowed to *add* vertices to $G$, it is possible to create a new spanning tree with less total weight than the original. The added nodes are called *Steiner points* and the resulting tree is called a *Steiner Minimal Tree (SMT).*[1,2,7,9] Figure 1 shows a comparison between a minimum spanning tree for a six-point problem and a Steiner minimal tree for the same problem.

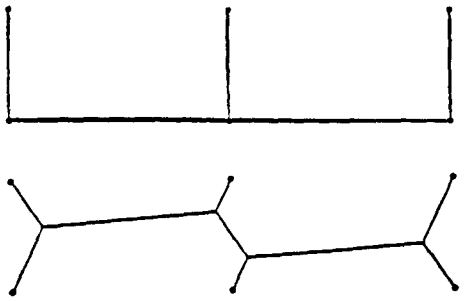

**Figure 1**: Consider the six vertices shown above. The vertical line segments are one unit long, the horizontal segments are each two units long. The minimum spanning tree on top has length 7. The Steiner minimal tree on the bottom for the same problem has length 6.616994.

Our first approach was to develop a parallel genetic algorithm (PGA) to solve this problem. For comparison, we also developed a traditional parallel branch-and-bound algorithm (PB&B) to exhaustively search the solution space. For small problem sizes, both PGA and PB&B performed quickly and consistently found the optimal solution. However, as the problem size and difficulty increased, PGA was less and less successful in finding the optimal solution and PB&B took longer and longer time. At this point, we sought to develop *hybrid algorithms*, algorithms which allowed each algorithm to capitalize on output of the other. The objective is to allow the algorithms to cooperate in order to accelerate overall performance.

A brief background to this problem is presented in Section II. The parallel genetic algorithm, the parallel branch-and-bound algorithm, and the three hybrid algorithms we developed are described in Section III. The results of the implementation of the algorithms are presented in Section IV. Conclusions and a discussion of future work are given in Section V.

## II. BACKGROUND

The idea of a genetic algorithm is credited to John Holland[8]. Holland considered the biological process of evolution at the chromosome level and felt that this process would be applicable to computational problems.[4] Holland began his work in the early 1970's and since then many others have followed his lead. Genetic algorithms have been applied to an assortment of problems including optimization problems like the one presented here[3,4,6].

A genetic algorithm approach to the Steiner tree problem is described by Kapsalis *et al.* [10]. Their process begins with a complete listing of points containing not only the original set of nodes S, but also possible Steiner points. The genetic algorithm is used to select a subset of the Steiner points that, when combined with the points in *S*, produces an MST. A binary chromosome is used in which 1's and 0's represent the presence or not, respectively, of nodes and Steiner points in the final solution. To ensure that all solutions contain the complete set *S*, a chromosome representing only the points in *S* is OR-ed with the population. Ten such chromosomes make up an initial population.

Kapsalis *et al.* use population replacement, more commonly called the generational model. Current solutions are selected and placed in a gene pool; solutions with above average fitness may be placed in the gene pool more than once. Solutions from these gene pools create an offspring pool of solutions through crossover and mutation. When the offspring pool has the necessary number of members, it replaces the previous population. This process of creating new populations is repeated until some specified stopping criterion is met.

Our approach differs from that of Kapsalis in two ways: (1) we work with Steiner subtrees, not the original points in the graph, and (2) PGA uses a steady-state model. Much information is available about the relationships of the Steiner subtrees among themselves, information which can be used to advantage in forming the final solution[7,14]. This will allow us to solve much larger problems.

## III. ALGORITHM

A Steiner minimal tree can be found through a two-step process: (1) creating a *T_list* of Steiner subtrees, and (2) extracting the final solution from the list. The T_list, named by Winter[14], consists of a large set of full Steiner subtrees formed using Steiner points. An extraction process retrieves a subset of these subtrees to form the complete Steiner minimal tree. *This paper focuses on the extraction process* which, as Harris notes, is significantly more time-consuming than the creation of the T_list for large problem sizes[7]. The T_lists used in this study were supplied by Harris.

### A. Genetic Algorithm

The approach used in this study is that of a parallel genetic algorithm (PGA). Each gene of a chromosome represents a single Steiner subtree of the T_list. The length of the chromosome is equal to the number of subtrees in the problem. The *ith* element of the chromosome is a 1 if the *ith* tree is included in the solution; otherwise, it is a 0. The chromosome 0010011000 indicates that of ten subtrees, only subtrees 3, 6, and 7 are present.

The basic genetic algorithm used in this study consists of six steps shown below:

1. Initialize the population of chromosomes.
2. Evaluate each chromosome in the population.
3. Create new chromosomes by mating current chromosomes.
4. Delete members of the population to make room for new chromosomes.
5. Evaluate the new chromosomes and insert them into the population.
6. If time is up, stop and return the best chromosome; if not go to 3.

Each chromosome also has a set of five flags used to identify invalid solutions. A solution is invalid if: (1) at least one graph node is not covered by a Steiner subtree, (2) at least two Steiner subtrees are incompatible, (3) at least one Steiner subtree is isolated, (4) the complete solution is not connected, or (5) the complete solution has cycles. Each of the restrictions has a predetermined penalty which is added to the overall cost of the solution. This increases the cost of poor solutions and causes them to be eliminated through the genetic process.

Selection is done using a linearly-biased random number generator[11]. Recombination is accomplished using *N*-point crossover[12], the natural extension of one-point crossover. Unlike one-point crossover, however, applying *N*-point crossover to two parents produces $2(N+1)$ children for $N>1$. For $N=2$, for example, if $a_1a_2a_3$ and $b_1b_2b_3$ are two parent chromosomes ready for crossover, it is possible to produce $2(2+1)=6$ children, namely $\{a_1b_2b_3,\ b_1a_2a_3,\ b_1a_2b_3,\ a_1b_2a_3,\ b_1b_2a_3,\ a_1a_2b_3\}$. In this study, *N* was typically 6, 7, or 8.

### B. Branch-and-Bound Algorithm

PB&B[13] performs a standard depth-first search of the solution space. Subtrees of the full search tree are distributed by a central processor in round-robin fashion to

as many other processors as are participating in the computation. Subtrees are of a pre-determined size, not so large as to cause an excessive amount of work to be performed by a small number of processors, yet not so small as to cause communication bottlenecks. The appropriate size was determined experimentally.

The complete search tree grows very quickly as the number of Steiner subtrees increases. Fortunately, it is not necessary to search the entire search tree; a few simple pruning rules drastically reduces the number of search tree nodes visited. A subtree of the search tree is not searched by PB&B if the root of that subtree meets any of the following criteria: (1) the solution is complete, (2) the cost of the prefix in the node is larger than the current best cost, (3) the partial solution is not connected, (4) two of the Steiner subtrees in the solution are incompatible, or (5) the partial solution has been encountered before.

### C. Hybrid Algorithms

#### 1. Extraction

PB&B must work through possibly very large search spaces. As it does, it gathers information about the partial solution it currently has and may abort a search down a path in the search space if it determines that the path cannot lead to a solution better than its current best.

To provide additional information to PB&B, we first experimented with a technique called *extraction*. We take the best three hundred solutions from one run of PGA and AND the bit string representations of the chromosomes. The result is a small set of genes (Steiner subtrees) common to all of the solutions. Because these were the best solutions in independent GA runs, the hypothesis is that these common subtrees have a high probability of being very good subtrees and are quite likely part of the optimal solution. Thus, PB&B modifies its search in a way that guarantees that these trees will be part of the final solution. Because of this restriction, it is possible that large portions of the search space will be pruned, thus speeding up the operation of PB&B.

#### 2. Recycling

The overall fitness of a randomly-generated initial population is rather poor because a randomly generated solution will often violate one or more of the requirements for a minimal Steiner tree and therefore incur a significant number of penalties. The population will require a great amount of change or growth to reach an optimal solution. This suggests a slight modification to the original genetic algorithm: the use of "recycled" solutions. A large pool of good solutions is available from any given run of PGA. From this pool, the best fifty to one hundred solutions may be chosen to seed a new population of size two hundred with the remaining solutions generated randomly. The genetic algorithm then runs exactly as before.

#### 3. Seeding

The flip-side of extraction is *seeding*. Here, solutions generated by PB&B are passed to PGA to provide additional diversity in its population. PGA may include these solutions immediately into its population or may save the solutions for a later recycling run. This approach is currently being tested; at the present time, no results are yet ready for presentation.

### D. Implementation

PVM, Parallel Virtual Machines[1], was used to implement parallel execution and communication. PVM is "...a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource[5]." PVM makes a network appear as a virtual machine. The process is started on each host machine. Host machines can be used in multiple overlapping PVM virtual machines.

After the creation of the virtual machine, PVM programs can be executed. The coding of PVM applications involves another part of PVM, a user library of routines. This library defines routines for spawning processes, interprocess communication, and other necessary parallel functions. Examples of functions available are *send(tid, msgtag)*, *recv(tid, msgtag)*, and *probe(tid, msgtag)*. In these functions *tid* refers to the task identifier assigned to a process by PVM during spawning and *msgtag* signifies a user defined message type. When a message is sent, it waits in the appropriate processor's mailbox until it is retrieved using *recv()*. A process can check its mailbox to see if a message has been received without actually receiving it by using *probe()*.

While PVM is designed so that it can be ported across a variety of machines, the only platform used in this implementation was the network of Sun workstations in the Department of Computer Science at Clemson University. The department Sun network consists of 4 Sun servers, 59 Sun-4 workstations, and 17 X terminals. Results were run over a collection of the Sun-4 workstations. The Sun-4 workstation utilizes a RISC technology, the Scaleable Processor Architecture (SPARC). Ethernet, RS-423, and RS-232C interfaces and protocols are used in data communication.

## IV. RESULTS

The results of runs on five different sets of Steiner subtrees are presented in this section. The number of subtrees are 58, 60, 71, 85 and 91. Each of these came from a different 100-node graph problem; the generation

[1] PVM was initially designed at Oak Ridge National Laboratory. It is distributed freely and information about receiving the necessary software can be requested by sending email to netlib@ornl.gov and including the message: "send index from pvm3".

| Size | PGA<br>10 procs | PB&B<br>8 procs | Optimal<br>Solution | Extraction<br>4 procs | Recycling<br>10 procs |
|---|---|---|---|---|---|
| 58 | 1.706792<br>11m | 1.705451<br>7m | 1.705451 | 1.705451<br>24m | 1.705451<br>60m |
| 60 | 1.458565<br>11m | 1.458565<br>3m | 1.458565 | 1.458565<br>12m | 1.458565<br>35m |
| 71 | 1.725388<br>13m | 1.723710<br>75m | 1.723710 | 1.723710<br>85m | 1.723710<br>80m |
| 85 | 1.638612<br>17m | 1.629994<br>300m | 1.629994 | 1.629994<br>251m | 1.629994<br>180m |
| 91 | 2.308022<br>20m | 2.300023<br>7 days | 2.300023 | 2.300023<br>2 days | 2.300023<br>300m |

**Table 1**

of the T_List of Steiner subtrees is performed separately (see Section III). The results are summarized in Table 1.

Column 1 lists the number of Steiner subtrees in the problem. Column 2 shows the results of the parallel genetic algorithm running on ten processors. Column 3 shows the results of the parallel branch and bound algorithm running on eight processors. The upper value in each table entry is the cost of the solution found; the lower value is the execution time expended, expressed in minutes unless otherwise specified. Column 4 gives the optimal solution for each of the problems. The last two columns show the results after applying the methods of Extraction and Recycling described in Section C.

First note that PGA, by itself, produced the optimal solution in only one of the five problems. These results were obtained after running between eight hundred and two thousand genetic algorithms for each problem size over a period of a year. The costs shown (e.g., 1.706792 for the problem with 58 subtrees) are the *best* obtained by PGA. Although never more than a few percent greater than optimal, PGA could not even always match its own best solution, and it failed to produce the *optimal* solution virtually all of the time. Although it did produce the optimal solution for the problem with 60 subtrees in five of the many hundreds of runs, the exercise, overall, was somewhat frustrating. One bright note about PGA, however, is the fact that it generates its solutions quickly, always within twenty minutes. If one is satisfied with *very good* solutions obtained right away, PGA can deliver.

On the other hand, the results of the PB&B runs are, by definition, optimal. PB&B searches the solution space exhaustively and, therefore, finds the optimal solution every time. PB&B, however, is impractical to use for very large problem sizes because execution times, can become quite large. The 71-tree problem runs for over an hour on eight processors. The 85-tree problem requires five hours. And the 91-tree problem runs eight processors for *seven days* before completion! (In fairness, PB&B's times of 75 minutes and 300 minutes for problem sizes 71 and 85 *do* represent a improvement over execution times of up to twelve hours reported previously[7].) In general, however, multiple-day execution times are not acceptable.

In summary, PGA provides us with *very good* solutions very quickly, but can rarely, if ever, provide optimal solutions. PB&B, on the other hand, provides optimal solutions all the time but only if you can wait for the algorithm to complete. Individually, neither algorithm is satisfactory. It was at this point that the authors tried a hybrid approach.

The results of recycling good solutions in PGA are shown in the rightmost column. Recall that in recycling, PGA collects good solutions produced in an earlier run and uses these as part of the initial population in the recycling run. The execution times shown represent *total* execution times which include the time to generate the initial good solutions and the time to run the final genetic algorithm which produces the optimal solution. Note that at the cost of longer execution times, PGA is able to produce the optimal solution *every* time. This experiment was conducted multiple times (at least five) for each problem size and the results were the same, i.e., PGA with recycling produced the optimal solution every time. Contrast this with PGA alone which failed to find the optimal solution in many hundreds of attempts.

For the two smaller problem sizes, 58 and 60, PGA with recycling is not cost-effective. PB&B is able to produce the same results much more quickly. For the 71-tree problem, PB&B with 8 processors is still more efficient than PGA with recycling using 10 processors. However,

for problem sizes 85 and 90, PGA with recycling is clearly superior to PB&B.

The results of *extracting* common Steiner trees from good solutions for use in PB&B are shown in the fifth column. The execution times represent *total* times which include the time required for the genetic algorithm to generate the initial solutions. Recall that the objective is for the PGA to provide clues regarding good solutions to PB&B in the hope that PB&B can use the information to prune away large portions of the search space. No execution time improvement is evident in the test on problem sizes 58 and 60. For problem sizes 71 and 85, PB&B with extraction using four processors is slightly more efficient than PB&B alone with eight. PB&B with extraction offers a significant improvement, however, in the 91-tree problem. Execution time was reduced from seven days with eight processors to two with four processors.

## V. CONCLUSIONS AND FUTURE WORK

This results presented here show much promise and allow us to envision a multiprocessor system working on a hybrid approach toward solving much larger problem sizes of the Steiner tree problem. Some of the processors are working on a version of PGA, others are working on a version of PB&B. PGA saves its solutions and recycles them periodically. The best solutions are set aside and the common Steiner subtrees are identified for use by PB&B. PB&B passes its solutions to PGA for immediate use in its current population or for future use in a recycling run. As PGA processors complete their assigned tasks, they may be instructed to restart with a new population of solutions. The process completes when the user is satisfied that either the search space has been searched exhaustively enough or that the solution obtained is judged sufficiently close to optimal.

The results presented here, however, are still quite preliminary and much more work is required before any firm conclusions can be drawn. We continue to work on testing these techniques; our goal is to work on much larger problems, on the order of 200 nodes and 150 Steiner subtrees and to experiment with *seeding* as well as recycling and extraction. The solution space of problems of this size are much too large for PB&B alone and much too vast for PGA alone. However, solutions to very large problems *can* be found if more tools such as extraction, recycling, and seeding can be developed to form new hybrid search algorithms.

## ACKNOWLEGEMENTS

The authors sincerely thank the reviewers for many helpful insights and suggestions.

## REFERENCES

[1] Cockayne, E.J. and D.E. Hewgill, Exact Computation of Steiner Minimal Trees in the Plane, *Information Processing Letters*, 22(3):151-156, March 1986.

[2] Courant, R. and H. Robbins, *What is Mathematics? An elementary approach to ideas and methods.* Oxford University Press, London, 1941.

[3] Davis, L., *Algorithms and Simulated Annealing*, Pitman Publishing, London, 1987.

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

[5] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sundram, *PVM 3 User's Guide and Reference Manual*, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1994.

[6] Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison/Wesley Publishing Company, Reading Massachusetts, 1989.

[7] Harris, F. Jr., *Parallel Computation of Steiner Minimal Trees*, PhD Dissertation, Department of Computer Science, Clemson University, Clemson, South Carolina, 1994.

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

[9] Hwang, F.K. and D.S. Richards, Steiner Tree Problems, *Networks*, 22(1):55-89, January 1992.

[10] Kapsalis, A., V.J. Rayward-Smith, and G.D. Smith, Solving the Graphical Steiner Tree Problem Using Genetic Algorithms, *Journal of the Operational Research Society*, 44(4):397-406, 1993.

[11] Pargas, R.P., *A Linearly-Biased Random Number Generator*, Technical Report 97-101, Department of Computer Science, Clemson University, Clemson, South Carolina.

[12] DeJong, K.A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, University of Michigan (Ph.D. Thesis) 1975, Dissertation Abstracts International 36(10) 5140B, University Microfilms Number 76-9381.

[13] Spoon, S.A. and R.P. Pargas, *A Parallel Branch and Bound Algorithm for Finding Steiner Minimal Trees*, Technical Report 97-102, Department of Computer Science, Clemson University, Clemson, South Carolina.

[14] Winter, P., An Algorithm for the Steiner Problem in the Euclidean Plane, *Networks*, 15(3):323-345, Fall 1985.