RC 19394 (84396) 1/28/94
Computer Science 30 pages

Research Report

Two Algorithms for Finding Optimal Communications Spanning Trees

Charles C. Palmer and Aaron Kershenbaum

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

Two Algorithms for Finding Optimal Communications Spanning Trees

Charles C. Palmer
Aaron Kershenbaum

IBM Research Division
Network Design Tools
Computer Science Department
T. J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

ABSTRACT: We consider two algorithms, one heuristic and one genetic, for finding solutions for the Optimal Communications Spanning Tree Problem (OCSTP). We first describe the basic problem and these two approaches to its solution. We then present computational experience with the algorithms, comparing them with one another. We show that while the heuristic is generally faster and produces results as good or better than the genetic algorithm (GA), the GA is more robust than the heuristic and is able to continue to produce reliably good solutions, without any modification to the basic procedure, even when the problem changes radically. The quality of the solutions produced by the heuristic, on the other hand, deteriorates as the underlying problem changes. Thus, the GA is important for validating heuristics and for helping to discover new ones as the new types of problems arise.

1. The Optimal Communications Spanning Tree Problem (OCSTP)

We are given a set, N, of n nodes, which are labeled with the numbers 1,2, ... , n. We are also given C_{ij}, the cost per unit capacity for a link between each pair of nodes i and j. We assume that C_{ij} is:

- symmetric
- strictly positive (for $i \neq j$)
- need not satisfy the triangle inequality.

We are also given R_{st}, the requirement between nodes s and t. We assume the requirements R_{st} are bidirectional; i.e., that they take up capacity both from s to t and from t to s. Thus, our network model is undirected. The links of the network, and hence its cost, can be thought of as a function of R_{st} for $s < t$. We assume that capacity is a continuous quantity; i.e. that the cost of a link of capacity A between i and j is simply $A \times C_{ij}$ for any value of A. Thus, we assume that all requirements (in both directions) between any pair of nodes can be combined into a single R_{st}.

We seek a tree (cycle-free graph) of minimum total cost satisfying a given requirement set. Thus we seek the tree T which minimizes

$$X(T) = \sum_{i,j} A_{ij}(T) \times C_{ij} \tag{1.1}$$

where $A_{ij}(T)$ is the capacity required in T for the link between node i and j.

Note that a requirement, R_{st}, will in general be satisfied by capacity on a path (sequence of links) between s and t. In the general case, we would be confronted by a routing problem; i.e., we would have to select the path each requirement follows. In this case, however, because we are dealing with trees (where there is, of course, a unique path between each pair of nodes), the routing problem disappears. We assume that each unit of requirement R_{st} takes up one unit of capacity on each link in the path between s and t. Thus,

$$A_{ij} = \sum_{\text{all } s,t\text{: link}(i,j) \text{ is in path between } s \text{ and } t} R_{st}$$

This problem is interesting for a number of reasons. First, it is an interesting network design problem in its own right. There are situations where the nature of

the nodes used in constructing a network limit the network topology to a tree. There are also cases where the actual link cost function (i.e., one with a large imbeded fixed cost) make a tree topology attractive.

The solution to this problem can also be used as a starting point for the solution to the general network design problem, where the topology is not constrained to be a tree and where the relationship between requirements and capacity is non-linear.

Finally, this problem is a member of an intractable class of combinatorial optimization problems which are not only NP-complete but also exhibit little "locality" in their solution space. Most classical approaches to optimization problems rely on the fact that good solutions are "close" to one another, and that one can move from one good solution to another by making small changes to the solution. These sorts of small changes include adding a link, deleting a link, or exchanging a small number of links for one another. In the best cases, the solution space and the objective function are convex and we are guaranteed a globally optimal solution even when using a local search technique.

In this problem, however, this incremental approach does not work. Adding a link to a tree makes a cycle. Deleting a link from a tree disconnects the network. Exchanging one link for another changes the flows (and hence the costs) on many links and may also introduce a cycle. It is therefore necessary to use techniques which are able to move between radically different solutions or else accept very local optima. We explore one algorithm of each kind in the discussion that follows and explore the strengths and weaknesses of each approach.

2. Heuristics for the OCSTP

Often, a star (i.e., all nodes directly connected to a single node) is a good solution to the OCSTP. This is particularly true when there is a significant fixed cost associated with the cost of the links; i.e., when the cost per unit capacity, C_{ij} is given by

$$C_{ij} = A + C'_{ij}$$

where A is a constant comparable to C'_{ij} for most i and j. Indeed, Hu [8] demonstrated that a star is a optimal solution to the OCSTP if the C_{ij} satisfy a stronger form of the triangle inequality:

$$C_{ij} < C_{ik} + a \times C_{kj} \tag{2.1}$$

for all k, where

$$a = \frac{N-2}{2N-2}$$

for a network with N nodes. Hu was able to show that if all C_{ij} satisfy equation 2.1 with respect to all k, then the optimal tree will have only one interior node; i.e., only one node of degree greater than one. The OCSTP is then reduced to the simple problem of finding the best center for the tree. This can be solved by evaluating all nodes as the center and choosing the one that leads to a minimum cost network.

A node i can be evaluated as a center in $O(N)$. To do so, one need only compute the sum

$$S_i = \sum_j C_{ij} W_j$$

where W_j is the total of all requirements between node j and all other nodes. Since the evaluation of W_j itself requires $O(N)$ effort, the evaluation of S_i is actually $O(N^2)$ if one includes the effort of evaluating W_j. However, W_j needs only be evaluated once for each j in order to be able to evaluate S_i for all i. Thus, the overall effort to evaluate S_i for all i is $O(N^2)$ and the problem of finding the optimal star is $O(N^2)$.

For many realistic problems, we have found in practice that the optimal star is the best solution we were able to find using any procedure. This is particularly true when the C_{ij} contain any significant fixed cost or when the R_{st} do not fall off rapidly as distance increases.

When a star is not optimal, the next most likely solution to try is a tree formed by connecting two stars together. Good solutions in this class can be found in $O(N^3)$ by examining all pairs of nodes as interior in the tree and associating all other nodes with the nearest (in cost) interior node.

It is, of course, possible to consider sets of three or more interior nodes as well, but the effort to do so for set of k interior nodes is $O(N^{(k+1)})$ and hence is not practical unless N or k remains small.

When a star is not a good solution (e.g., when the requirements exhibit a strong community of interest with large requirements between nodes close to one another), another heuristic starts with a minimum spanning tree (MST) and then considers local exchanges to eliminate interior nodes. This starts with a solution which is very "unstarlike" and moves towards a star. Thus, we explore a completely different part of the solution space. If this heuristic also converges to a star, we might optimistically hope that the star is in fact a very good (if not optimal) solution. If it does not converge to a star, then we have the opportunity to see solutions which are very

different form stars in situations where stars are not good solutions. A more detailed description of this heuristic follows.

1. Choose a network center (median) M. This can be, for example, the center of the best star in the first heuristic described above.

2. Find an MST, T, on the set of nodes. Compute, $S(T)$, the weighted (by the size of the requirements) sum of the lengths of the paths between all pairs of nodes. Let T be the current tree. $S(T)$ is the best sum found so far.

3. Working outward from M, for each node i where i is not itself M and $P[i]$, the predecessor of i in the current tree, is not M:

   (a) Let PP be the predecessor of $P[i]$.

   (b) Consider making PP the predecessor of i. Call the tree so formed T'.

   (c) Evaluate $S(T')$.

   (d) If $S(T') < S(T)$, set $T = T'$ and $P[i] = PP$; i.e., let PP be the predecessor of i.

   (e) If PP is not M, consider node i again.

Thus, we consider moving nodes up the tree, making their predecessors closer and closer to the median M, as long as doing so improves the tree. It is possible that this will form a star or, altrenatively, that it will not in the the case where there are groups of nodes which are close to one another and which have a strong community of interest; i.e., large requirements between them.

For example, suppose we have a problem with six nodes. We let $R_{ij} = 1$ for all $i \neq j$ and define the costs as

$$C_{ij} = \begin{matrix} 0 & 3 & 1 & 4 & 2 & 4 \\ 3 & 0 & 4 & 3 & 5 & 5 \\ 1 & 4 & 0 & 5 & 3 & 5 \\ 4 & 3 & 5 & 0 & 5 & 3 \\ 2 & 5 & 3 & 5 & 0 & 2 \\ 4 & 5 & 5 & 3 & 2 & 0 \end{matrix}$$

Then, letting each node in turn be the median we find

$$\begin{aligned} M = 0 &\Rightarrow S = 70 \\ M = 1 &\Rightarrow S = 100 \\ M = 2 &\Rightarrow S = 90 \\ M = 3 &\Rightarrow S = 100 \\ M = 4 &\Rightarrow S = 85 \\ M = 5 &\Rightarrow S = 95 \end{aligned}$$

So, the best choice is $M = 0$ with $S = 70$. Now we form an MST to see if we can improve on this. In this example, there are several possible MST's, all with a total link length of 11. We choose (at random) the following MST, T, for which $S(T) = 70$:

$$(0,2),(0,4),(4,5),(0,1),(1,3)$$

If we consider T as a tree rooted at the median, node 0, we have the following predecessors:

i	0	1	2	3	4	5
$P[i]$	0	0	0	1	0	4

Working outward from the median node, we consider node 3, one of the nodes whose predecessor is not the median. Its predecessor, P, is node 1. PP, the predecessor of P, happends to be the median node. We consider setting $P[3] = PP = 0$. When we do, the new tree T' has $S(T') = 66$, an improvement, so we set $P[3] = 0$. Since $P[3]$ is now the median, we cannot push node 3 up any further. We next consider pushing up $P[5]$, another node whose predecessor is not the median. In this case, $PP = 0$. We consider setting $P[5] = 0$. Doing so forms the star at node 0, which we already know has $S = 70$. So, we do not make this exchange. There are no further nodes to consider, so we stop with the tree

$$(0,2),(0,4),(4,5),(0,1),(0,3)$$

which has $S = 66$.

3. A Genetic Algorithm for the OCSTP

A genetic algorithm [6][7][4], or GA, produces new (and, hopefully better) solutions to a problem through the recombination of other solutions. Specifically, given two solutions, S_1 and S_2, to a problem, we can produce a new solution by taking part of S_1 and part of S_2. The exact way a GA combines different solutions to produce

new ones is a function of how solutions are represented within the GA. In this section we give a brief background on genetic algorithms and then discuss the representation of trees in genetic algorithms and how the genetic algorithm parameters should be set for the OCSTP.

3.1. Background

In the traditional GA approach, a solution is represented by a chromosome which is comprised of genes which describe the components of the solution in much the same way that chromomsomes represent specific traits of organisms in nature. In a GA, an offspring is produced by applying a "crossover" operator to a pair of parents which results in the offspring having some genes from each. Specifically, if a solution is represented by a vector

$$X = \{x[i], i = 1, 2, \ldots N\}$$

then two (parent) solutions X_1 and X_2 can produce two offspring X_3 and X_4 through simple (one-point) crossover by the rule

$$X_3 = \{x_3[i], i = 1, 2, \ldots N\}$$
$$X_4 = \{x_4[i], i = 1, 2, \ldots N\}$$

where

$$\begin{aligned} x_3[i] &= x_1[i] \quad , i \leq k \\ x_3[i] &= x_2[i] \quad , k < i \leq N \end{aligned}$$

$$\begin{aligned} x_4[i] &= x_2[i] \quad , i \leq k \\ x_4[i] &= x_1[i] \quad , k < i \leq N \end{aligned}$$

Thus, offspring are produced by selecting a crossover site k in the chromosome and taking all genes to one side of the crossover site from one parent and all the genes to the other side from the other parent. The order of the parents makes no difference. The selection of parents is a random selection from a population of chromosomes, weighted by the quality, or *fitness*, of the chromosomes. After crossover is performed, each of the offspring's genes may be changed further through mutation wherein the value of a gene is changed with some typically very low probability.

There are, in general, many possible representations for solutions to a given problem. In our case, the representation, or encoding, of trees must posses certain properties:

1. It should be capable of representing all possible trees.

2. It should be unbiased in the sense that all trees are equally represented; i.e., all trees should be represented by the same number of encodings. This property allows us to effectively select an unbiased starting population for the GA and gives the GA a fair chance of reaching all parts of the solution space.

3. It should be capable of representing only trees. To the extent that non-trees can be represented, it becomes more difficult to generate a random initial population for the GA. Worse yet, it becomes possible for crossover and mutation to produce non-trees from valid parent trees.

4. It should be easy to go back and forth between the encoded representation of the tree and the tree's representation in a more conventional form suitable for evaluating the fitness function and constraints.

5. It should possess locality in the sense that small changes in the representation make small changes in the tree. This allows the GA to function properly by having the encoding truly represent the tree. Thus, when crossover takes place, parts of the parent trees are inherited and good traits can propagate from one generation to the next. Without this property, the GA tends to drift rather than converge to a highly fit population.

Ideally the representation of a tree for a GA should have all of these properties. Unfortunately, most representations trade some of these desirable traits for others. In the following sections, we will discuss a number of tree representations and evaluate them on the basis of how well they meet these criteria.

3.2. Comparison of Existing Tree Representations

We are given a set of n nodes, N, which are labeled with the numbers 1,2, ... , n. In any specific problem, we are also given characteristics of the edges between the nodes such as their lengths. Here, however, we are concerned only with the representation of the trees themselves and so we ignore the edge properties.

We denote the (undirected) edge between nodes i and j by (i, j). We assume here that all edges are possible candidates for inclusion in the tree; i.e., that the underlying graph from which the tree is formed is a complete graph. Note that it is always possible to transform a problem on a given graph into one on a complete

graph by adding auxiliary edges to the graph with sufficiently undesirable properties such as very long lengths.

It is possible to give an orientation to the edges in a tree by designating one node as the root and having all edges oriented towards (or, alternatively, away from) this node. Such trees are called rooted trees. Sometimes, the nature of the problem makes this designation significant; e.g., in the case of a tree of shortest paths to (or from) a given node. Sometimes the problem specification fixes the identity of the root. Other times its selection is part of the problem. In all cases, it is possible to represent an arbitrary tree as a rooted tree if we so desire. We will do so here.

It has been shown [3] that the number of possible trees in a complete graph on N nodes is $N^{(N-2)}$. Since each such tree can correspond to N possible rooted trees, with any node designated as the root, there are thus $N^{(N-1)}$ possible rooted trees. We can thus assess how efficient any representation is by comparing the number of graphs that can be represented by it to the possible number of trees. If the former is much larger than the latter, many non-trees can also be represented and this is, as we mentioned above, a problem.

3.2.1. Characteristic Vector. If we associate an index k with each link (i, j), we can represent a tree T as a vector $E = e_k$, k =1, 2, ... K , where K is the number of edges in the underlying graph and e_k is one if edge k is part of T and zero otherwise.

In a complete graph, $K = N(N-1)/2$. There are thus $2^{(N(N-1)/2)}$ possible values for E and, unfortunately, most of these are not trees. Indeed, since all trees have exactly $N-1$ edges, if E contains other than $N-1$ ones it is not a tree. Even if E contains exactly N ones, it is unlikely that it represents a tree. Indeed, the probability of a random E being the representation of a tree is infinitesimally small as N increases. It is, in fact, of order $2^{-[N(\frac{N}{2}-\log 2(N))]}$.

Thus, if we were to generate random vectors in order to provide a starting population for a GA, it is quite likely that none of them would be trees. Furthermore, when we mate any two trees in the course of a GA, it is very likely that none of the offspring would be trees.

It is an $O(N^2)$ effort to go back and forth between this encoding and a tree. This is not very good, since as we will see there are other methods where only $O(N)$ effort is required.

On the positive side, all trees can be represented by such vectors, all are represented equally (once), and the representation does possess a natural locality; chang-

ing a bit in the vector adds or deletes a single edge. On the whole, however, this is a poor representation for a GA because of the extremely low probability of obtaining a tree.

3.2.2. Predecessors. An alternative representation is to designate a root, r, for the tree and then record the predecessor of each node in the tree rooted at r. Thus $Pred[i] = j$ if j is the first node in the path from i to r in T. Thus, every rooted tree T is represented by a unique N "digit" number, where the "digits" are numbers between 1 and N. Equivalently, we could say that every (unrooted) tree is represented by N of these numbers. We see, therefore, that this encoding is unbiased and covers the space of solutions.

There are N^N such numbers. Since there are $N^{(N-2)}$ trees, a random number of this type represents a tree with probability $\frac{1}{N}$. This is a great improvement over the characteristic vector, but still allows for many non-trees being generated both in the initial population and during the breeding which takes place during the course of the GA.

Given a matrix mapping node pairs into edges, which can be set up at the start of the GA and required $O(N^2)$ space, we can transform back and forth between this representation and a list of edges in $O(N)$. This is an improvement over the characteristic vector representation. Thus, we see that, at least for complete graphs, this representation is significantly better than the one above. For sparse graphs, however, the improvement diminishes.

3.2.3. Prüfer Numbers. A third possible encoding is the Prüfer number [9] associated with a tree, defined as follows. Let T be a tree on N nodes. The Prüfer number, $P(T)$, is an $N-2$ "digit" number, where once again the digits are numbers between 1 and N and are defined by the following algorithm. We assume N is at least two. Indeed, for the algorithm to be non-trivial, N should be at least three.

1. Let i be the lowest numbered leaf (node of degree 1) in T. Let j be the node which is the predecessor of i. Then j becomes the rightmost digit of $P(T)$. We build $P(T)$ by appending digits to the right; thus, $P(T)$ is built and read from left to right.

2. Remove i and the edge (i, j) from further consideration. Thus, i is no longer considered at all and if i was the only successor of j, then j has become a leaf.

3. If only two nodes remain to be considered, stop. $P(T)$ has been formed. If more than two nodes remain, return to Step 1.

As an example of how this algorithm works, consider the tree shown in Figure 1. Node 2 is the lowest numbered leaf and node 3 is its predecessor. We therefore select 3 as the first digit of $P(T)$. We then remove node 2 and the edge $(2,3)$ from consideration and node 4 becomes the lowest numbered leaf. So, the next digit of $P(T)$ is also a 3 since 3 is also the predecessor of node 4. We now remove node 4 and node 3 itself becomes a leaf and, in fact, the lowest numbered leaf. Node 1, its predecessor is thus the next digit of $P(T)$. Node 5, with predecessor 1, becomes the lowest numbered leaf and the next digit of $P(T)$ is again 1. There are now only two nodes left to be considered, and so we stop with $P(T) = 3311$.

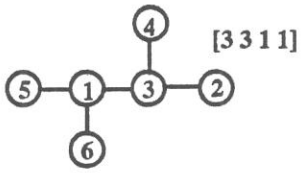

Figure 1: A Tree and its Prüfer number.

It is also possible to go from a Prüfer number to a unique tree via the following algorithm.

1. Let $P(T)$ be the original Prüfer number and let all nodes not part of P(T) be designated as eligible for consideration.

2. If no digits remain in $P(T)$, there are exactly two nodes, i and j, still eligible for consideration. (This can be seen by observing that as we remove a digit from $P(T)$ in Step 3 below, we remove exactly one node from consideration and there are $N-2$ digits in the original $P(T)$). Add (i,j) to T and stop.

3. Let i be the lowest numbered eligible node. Let j be the leftmost digit of $P(T)$. Add the edge (i,j) to T. Remove the leftmost digit from $P(T)$. Designate i as not no longer eligible. If j does not occur anywhere in what remains of $P(T)$, designate j as eligible.

4. Return to Step 2.

In the example given in Figure 1, we start with $P(T) = 3311$ and with nodes $2, 4, 5$ and 6 eligible. Node 2 is the lowest numbered eligible node. Node 3 is the leftmost digit of $P(T)$. We thus add $(2, 3)$ to T, remove 2 from further consideration and remove the leftmost digit of $P(T)$ leaving $P(T) = 311$. Node 4 is now the lowest eligible node and the leftmost digit of what remains of $P(T)$. We thus add $(4, 3)$ to T, remove 4 from further consideration and remove the second 3 from $P(T)$. Node 3 is now no longer part of $P(T)$ and becomes eligible. Indeed, it is the lowest such number. We thus add $(3, 1)$ to T, remove the leftmost 1 from $P(T)$ and remove 3 from further consideration. Now $P(T) = 1$ and only nodes 5 and 6 are eligible. We therefore add $(5, 1)$ to T, remove the last digit of $P(T)$ and designate 5 as not eligible. Node 1 is now eligible, since it is no longer part of $P(T)$. $P(T)$ is now empty and only nodes 1 and 6 are eligible. Thus we add $(1, 6)$ to T and stop. The tree in Figure 1 has been formed.

There are $N^{(N-2)}$ Prüfer numbers for a graph with N nodes. This is exactly the number of trees possible in such a graph. There is, in fact, a one to one correspondence between trees and Prüfer numbers, the transformation being unique in both directions.

Thus, Prüfer numbers are unbiased (each tree is represented once), they cover the entire space of trees, and they do not represent anything other than trees. The transformations back and forth between edges and Prüfer numbers can be carried out in $O(N \log N)$ with the aid of a heap. The algorithms are, as we see above, somewhat more complex intellectually than those for the two preceding representations, but this is not a serious disadvantage.

The real disadvantage of this representation is that is has relatively little locality. While any offspring formed by taking parts of two Prüfer numbers will indeed be a tree, it need not resemble the parent trees at all. Indeed, changing even one digit of a Prüfer number, can change the tree dramatically. Consider, for example, the six node trees formed from the Prüfer numbers 3241 and 3242 which have only two of their five edges in common.

We thus see that while each of these representations is acceptable by some of the criteria listed above, none is entirely adequate and it would be desirable to find a more suitable representation for trees within Genetic Algorithms. We present such a representation in the following section.

3.3. Two New Representations for Trees

We set off to design a new representation for trees that would satisfy the criteria given in section 3.1. Our first representation was based on permutations of predecessor vectors and was rather elegant, but we eventually found it to have little locality and a genetic algorithm using it performs poorly[1]. Our second representation applied evolving edge and node weights to drive the genetic algorithm towards a solution. This latter representation proved to be far superior to any we had used before in all respects. We describe both of these representations here because the shortcomings of the former are just as instructive as the advantages of the latter.

3.3.1. "Right Hand Rule" Representation. The predecessor representation of rooted trees has many desirable properties. It is reasonably compact. There is a natural transformation between the encoding and the representation of the tree as a set of edges. It covers the entire space of solutions in an unbiased way. It is reasonably local in that changing one predecessor changes just one edge in the tree. Its major drawback is that it is capable of representing non-trees. Specifically, a random N digit number has a probability of only $\frac{1}{N}$ of representing a tree.

Using such a representation in a GA decreases the effectiveness of the GA dramatically. To make use of it we must either accept a large percentage of non-viable members in the population or else we must come up with rules and constraints which make the production of non-trees less likely. The latter do not just complicate the implementation of the GA. They undermine its validity by potentially introducing a bias into the breeding process. This can result in an incomplete search of the solution space and the loss of one of the major strengths of the GA approach.

It is possible, however, to modify the predecessor representation to completely eliminate non-trees without destroying any of its other desirable properties. By doing so, we obtain a representation which satisfies all of the criteria given in section 3.1.

The key to effectively modifying the predecessor representation is to see what can go wrong with the original one. Any tree, T, can be represented by a vector of predecessors, $P(T)$, which can also be thought of as an N digit number. Again, each digit is a number between 1 and N. Now, however, there are N digits instead of $N-2$. The rule used for transforming a predecessor "number" to a tree is much

[1] After designing this representation we discovered Davis' [1] work on using permutations as a crossover-proof representation. The problem addressed in his work was apparently more amenable to this scheme.

simpler than the rule for Prüfer numbers: if the i-th digit of $P(T)$ is j, add edge (i, j) to T.

By convention, we set the r-th digit of $P(T)$ to r when the root of the tree is r and we ignore this digit in adding edges to T. Thus, any tree can be represented by N possible $P(T)$'s by selecting each of the N nodes as the root. Three of the possible $P(T)$'s for the tree in Figure 1 are 131311, 322311, and 631316, which correspond to making the root nodes 1, 2, and 6, respectively.

There are, however, N digit numbers which do not correspond to any tree. Consider, for example, the number 114635. This corresponds to a graph with the cycle 34653 in it and could be formed by initially selecting random digits or by breeding valid trees (e.g., by taking the leftmost five digits of 114631 and the rightmost one digit of 555555).

So, the problem is that it is possible for the predecessors to include one or more cycles. We can avoid this problem, however, by ordering the nodes and then only allowing a node to choose its predecessors from among the nodes to its right. It is clear that if we restrict ourselves to selecting predecessors in this way that all edges will "point to the right" and no cycles can be formed.

Any tree, and only trees, can be characterized by the following rules:

1. Let $S_1(T)$ be an ordering of the nodes; i.e., a permutation of the integers from 1 to N.

2. Let $S_2(T)$ be an $N-1$ digit number such that all the digits are between 2 and N and the i-th digit is strictly greater than i. The i-th digit of $S_2(T)$ gives the position (in the ordering given by $S_1(T)$) of the node which is the predecessor of i in T.

For example, one possible representation of the tree in Figure 1 is $S_1(T) = 651432$ and $S_2(T) = 33556$. We interpret this encoding as follows:

- Node 6 is the first node in the ordering and its predecessor is the third node in the ordering (node 1),

- Node 5 is the second node in the ordering and its predecessor is the third node in the ordering (node 1), etc.

Any tree is representable in this way. In fact, any tree is representable in many ways. First, the above representation is for rooted trees, so any tree can be represented in at least N ways, considering each of the nodes as the root. But this is not all. $S_1(T)$ is a permutation of N numbers. Thus, $S_1(T)$ can take $N!$ values. The leftmost digit of $S_2(T)$ can take $N-1$ values; The next leftmost digit can take $N-2$ values; etc. So, $S_2(T)$ can take $(N-1)!$ values. Each combination of S_1 and S_2 gives rise to a tree, and no combination of S_1 and S_2 gives rise to anything other than a tree since the interpretation of S_2 guarantees that there are no cycles.

Given a specific tree, T, we can select a node as the root and then postorder the nodes[2]. This postordering gives is what we have been calling $S_1(T)$. A given postordering, $S_1(T)$, uniquely specifies $S_2(T)$. Conversely, given any specific S_1 and S_2, it is easy to form a specific tree T.

Unfortunately, different rooted trees have different numbers of postorderings associated with them. A chain rooted at one end has only one. A star rooted at the center has $(N-1)!$. Indeed, different rooted trees corresponding to the same tree can have a different number of postorderings associated with them. The same chain mentioned above, this time rooted at a node at the center of the chain, has

$$\frac{(N-1)!}{((\frac{(N-1)}{2})!)^2} = \binom{N-1}{\frac{(N-1)}{2}}$$

postorderings associated with it, for N odd.

Since there are $N^{(N-2)}$ possible trees and $N!(N-1)!$ possible values for S_1 and S_2, the average number of times a tree is represented by an S_1,S_2 combination is

$$\frac{N!(N-1)!}{N^{(N-2)}}$$

Note that this quantity is not in general an integer.

So, this representation is biased in that some trees are represented more times than others. Fortunately, we can calculate the number of times each tree is represented and compensate for the bias.

One way of doing this is to observe that a given tree T can not be obtained from just any permutation. It must be a permutation which represents a postordering of the nodes of T. Since $S_1(T)$ uniquely specifies $S_2(T)$ for a given T, the number of

[2] A postorder of the nodes is any ordering which places predecessors to the right of their successors. In general, there are many postordering corresponding to a given rooted tree.

times T is representable by an S_1,S_2 combination is exactly equal to the number of permutations of the numbers from 1 to N which represent postorderings of T.

Theorem 1. *Let T be a rooted tree on N nodes. Let N_i be the number of nodes in the subtree rooted at node i, where i is any node in T. Then the number of postorderings of the nodes in T is given by:*

$$\frac{N!}{\prod_j N_j}$$

Proof. . The proof is by induction on the number of nodes in the tree. There is only one tree on two nodes and this tree satisfies the conditions of the theorem. Suppose that all trees with fewer than N nodes satisfy the conditions of the theorem. Now consider a tree, T, with N nodes. There are two cases to consider.

Case 1: The root of T has only one subtree, T'. In this case, every postordering of T is just a postordering of T' with the root of T appended to the end of the postordering, and the number of postorderings of the nodes of T is the same as the number of postorderings of the nodes of T'. This satisfies the conditions of the theorem since, in going from T' to T, the numerator is multiplied by N (as we go from $(N-1)!$ to $N!$) and the denominator is also multiplied by N (as we add an N to the product to account for the new (sub)tree, T, with N nodes in it. Thus, the number of subtrees in T satisfies the theorem.

Case 2: The root of T has two or more subtrees. For simplicity, let us first consider the case of exactly two subtrees. Let T_1 be one of these subtrees and let T_2 be the other subtree. Both T_1 and T_2 have fewer than N nodes and so, by the induction hypothesis, both satisfy the theorem. Let $N(A)$ be the number of nodes in an arbitrary tree A and let $NP(A)$ be the number of postorderings of the nodes of A. Any postordering of the nodes of T can be thought of as a postordering of the nodes of T_1 interleaved with a postordering of the nodes of T_2. The number of ways of interleaving nodes of T_1 and T_2 is equal to:

$$\frac{(N(T_1)+N(T_2))!}{(N(T_1))!\,(N(T_2))!}$$

Since $N(T_1)+N(T_2)=N-1$, this is $\binom{N-1}{N(T_1)}$, or

$$\frac{(N-1)!}{(N(T_1))!\,(N(T_2))!}$$

Within each interleaving, there are $NP(T_1)$ ways of arranging the nodes of T_1 and $NP(T_2)$ ways of arranging the nodes of T_2. By the induction hypothesis,

$$NP(T_i) = \frac{(N(T_i))!}{\prod_{j \in T_i} N_i}$$

where the product is taken over all subtrees in T_i, for $i = 1, 2$. Thus the total number of postorderings of the nodes of T is the product of these three factors

$$\frac{(N-1)!}{(N(T_1))!\,(N(T_2))!} \times \frac{(N(T_1))!}{\prod_i N_i} \times \frac{(N(T_2))!}{\prod_i N_i} = \frac{N!}{\prod_i N_i}$$

The denominator in the last expression is the product taken over all subtrees in T and is obtained by observing that the subtrees of T are precisely those of T_1 plus those of T_2 plus T itself (with N nodes).

A similar argument can be made when the number of subtrees is more than two. In this case we have corresponding products and terms for each subtree, T_i, of T for an arbitrary i. □

Knowing the number of postorderings, we know the number of representations for each tree. It is thus possible to compensate for the different number of times each tree is represented (e.g., by keeping a tree with probability equal to the reciprocal of this number), and thus obtain an unbiased set of trees. The effort in computing this factor is $O(N)$.

We now turn to the problem of generating the S_1,S_2 combination in a way that maintains locality. One way of doing this is to associate a number with each node. If we sort these numbers, say smallest first, we obtain a permutation. If we associate two numbers with each node, we can obtain two permutations. The first permutation is used directly to obtain S_1. The second is used to obtain S_2 using the following algorithm:

1. Let X be the identity permutation on the integers from 1 to $N-1$. Let P be an arbitrary permutation of the integers from 1 to $N-1$.

2. For $i = 1$ to $N-1$ do Step 3

3. Let j be $P[i]$, the number in position i of P. Let k be position that j currently occupies in X. Set $S_2[i] = i + k$. Exchange j and $X[N-i]$ in X.

As an example, suppose that N is 6 and we generate 6 random numbers: 37, 90, 78, 61, 32, 19. Sorting these from smallest to largest, we get 19, 32, 37, 61, 78, 90 and the permutation 651432 (from the original positions of the numbers currently occupying each of the positions in the sorted sequence). Thus, $S_1 = 651432$. Next we generate 5 random numbers: 40, 29, 86, 68, 67 Sorting these smallest first yields the permutation $P = 21543$ We now generate S_2 using the above algorithm:

```
i = 1; j = P[i] = 2; k = 2 since X[2] = j;
  S2[1] = i + k = 3
  Exchange X[2] and X[5]: X = 15342
i = 2; j = P[i] = 1; k = 1 since X[1] = j;
  S2[2] = i + k = 3
  Exchange X[1] and X[4]: X = 45312
i = 3; j = P[i] = 5; k = 2 since X[2] = j;
  S2[3] = i + k = 5
  Exchange X[2] and X[3]: X = 43512
i = 4; j = P[i] = 4; k = 1 since X[1] = j;
  S2[3] = i + k = 5
  Exchange X[1] and X[2]: X = 34512
i = 5; j = P[i] = 3; k = 1 since X[1] = j;
  S2[3] = i + k = 6
  Exchange X[1] and X[1]: X = 34512
```

Thus, the permutation $P = 21543$ yields $S_2 = 33556$.

If we carefully implement the algorithm to generate S_2, keeping the inverse permutation of X (i.e., keeping track of which number is in each position and in what position each number is), the algorithm is $O(N)$. Thus, we can transform back and forth between the tree and the encoding in $O(N)$, which is the best complexity possible. Another feature of this encoding is that the higher values of S_2 correspond to "star-like" trees while the lower values correspond to "chain-like" trees.

Genetic Algorithm Control Parameters. Because genetic algorithms had not been applied to the OCSTP or this encoding before, we had to determine what values of the control parameters would be appropriate. To do this, we followed the example of Grefenstette [5] and built a *meta-GA* to search the space of control parameters. A chromosome in the *meta-GA* contained particular values for the five genetic algorithm control parameters. The fitness of a *meta-GA* chromosome

was determined by averaging the best solutions found by using the parameters it contained in several runs of our genetic algorithm for the OCSTP. To resolve ties, we added a small bias to the returned fitness value to represent the running time of the genetic algorithm so that quicker runs would have a slight advantage. We used the "standard" genetic algorithm parameters defined by DeJong [2] for the *meta-GA* itself. In each of these runs, we limited the total number of evaluations to 10,000 and we allowed the meta-GA to vary the control parameters through these ranges:

Population size 5 to 1005 by 10

Crossover rate 40% to 100% by 0.6%

Mutation rate 0.01% to 10.1% by 0.1%

Generation Gap 30% to 100% by 0.7%

Scaling 1 to 5 by 0.04

The Cartesian product of these five parameters defines a space of 10^{10} genetic algorithms that was searched by the meta-GA. By allowing the meta-GA to vary the population size while limiting the total number of evaluations to 10,000, the population parameter also specified the number of generations that our genetic algorithm would run since $generations = \frac{evaluation\_budget}{population\_size}$. These runs were made on a set IBM RiscSystem/6000 systems and had running times from three to fifteen days. The *GAucsd 1.4* [11] tool was used for all of our genetic algorithm experiments.

We ran the meta-GA for a set of sample problems having 6, 12, 24, 47 and 98 nodes. In these problems, the requirements between cities (nodes) were made inversely proportional to their distance and the link costs were actual tariff rates. These produced several combinations of "good" parameters, the best 5% of which are summarized in table 1. These runs produced mixed results in that some of the resulting parameters were in reasonable agreement and others were not. We were further disappointed to see that GAs using these parameters produced the poor solutions to the OCSTP shown in table 2. In trying to understand why this was happening, we realized that the encoding wasn't able to maintain good locality. The wide separation of the predecessors on the left of the chromosome and the permutation on the right led to schemata of rather high length. As discussed earlier, dependence upon long schemata has a negative effect on the performance of a genetic algorithm and can produce the kind of mediocre results we encountered.

# of Nodes	Population Size	Crossover Rate	Mutation Rate	Generation Gap	Scaling Constant
(min)	5	0.400	0.0001	0.300	1.00
$6(best)$	25	0.820	0.0381	0.727	4.12
(max)	995	0.994	0.0991	0.993	4.96
(min)	35	0.490	0.0101	0.825	1.00
$12(best)$	55	0.790	0.0111	0.902	1.00
(max)	115	0.910	0.0121	0.993	1.40
(min)	125	0.520	0.0001	0.839	1.00
$24(best)$	135	0.910	0.0001	0.972	1.56
(max)	175	0.910	0.0021	0.993	2.88
(min)	95	0.850	0.0011	0.832	1.04
$47(best)$	105	0.868	0.0011	0.986	1.12
(max)	105	0.892	0.0021	0.993	1.48
(min)	105	0.850	0.0001	0.853	1.24
$98(best)$	135	0.988	0.0001	0.860	1.24
(max)	135	0.988	0.0001	0.860	2.24

Table 1: Meta-GA Results

technique	6	12	24	47	98
GA	1,386,360	7,035,895	45,712,100	192,500,000	1,800,810,000
Heuristic	1,386,360	7,134,530	37,952,000	158,612,156	780,999,474

Table 2: Comparison of "right hand rule" GA and Heuristic results

3.3.2. "Link and Node Bias" Encoding. We began anew by approaching the problem from a different direction. Experience has shown that for a given problem (nodes, requirements, and costs) certain nodes should be interior nodes and others should be leaf nodes. With this in mind, we designed a new encoding that allowed the genetic algorithm to search for nodes with these tendencies while looking for solutions to the OCSTP. In this encoding, the chromosome holds a *bias* value for each node. For example, in a four node problem the chromosome would contain four biases $[b_1 b_2 b_3 b_4]$. These values are multiplied by a parameter, P, and by the maximum link cost, $C_{\max}$, and are added to the cost of all links that have the node as an endpoint. The cost matrix would be biased by these values using

$$C'_{ij} = C_{ij} + PC_{\max}(b_i + b_j)$$

The tree that the chromosome represents is then found by applying Prim's algorithm [3] to find a minimal spanning tree (MST) over the nodes using the biased cost matrix. Finally, this MST is evaluated using the original cost matrix to determint the tree's fitness for the OCSTP.

This seemed sufficient at first, but we found that it didn't cover the space of all trees in certain cases. To see this, consider the network shown in figure 2. Suppose

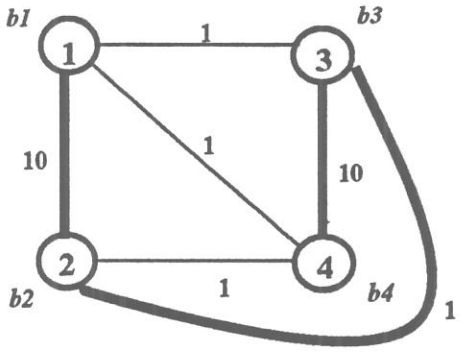

Figure 2: Example of a tree not representable using only node biases.

we want to encode the tree having links $(1,2)(2,3)(3,4)$. It is well known that a MST cannot contain the longest edge in any cycle. Thus, if $(1,2)$ is to be part of the MST, it must be true that

$$C'_{12} \leq C'_{13} \tag{3.1}$$

$$C'_{12} \leq C'_{23} \tag{3.2}$$

since $(1,2)(2,3)(1,3)$ form a cycle. Similarly,

$$C'_{12} \leq C'_{14} \tag{3.3}$$

$$C'_{12} \leq C'_{24} \tag{3.4}$$

Also, if $(3,4)$ is to be part of the MST, it must be true that

$$C'_{34} \leq C'_{13} \tag{3.5}$$

$$C'_{34} \leq C'_{14} \tag{3.6}$$

$$C'_{34} \leq C'_{23} \tag{3.7}$$

$$C'_{34} \leq C'_{24} \tag{3.8}$$

But, these inequalities contain four contradictory pairs. For example,

$$\begin{aligned} 3.1 \Rightarrow \quad & C'_{12} \leq C'_{13} \\ & 10 + b_1 + b_2 \leq 1 + b_1 + b_3 \\ & b_2 + 9 \leq b_3 \end{aligned} \tag{3.9}$$

$$\begin{aligned} 3.8 \Rightarrow \quad & C'_{34} \leq C'_{24} \\ & 10 + b_3 + b_4 \leq 1 + b_2 + b_4 \\ & b_3 + 9 \leq b_2 \end{aligned} \tag{3.10}$$

Thus, there is no choice of $[b_1 b_2 b_3 b_4]$ that would result in the links $(1,2)(2,3)(3,4)$ being selected as the MST using the given link costs. Such a tree is clearly not useful as a solution to the OCSTP, but in the interest of producing a generally useful tree representation we modified the representation to include link biases as well.

In this representation, the chromosome has biases for the N nodes and each of the $\frac{N(N-1)}{2}$ links, for a total of $\frac{N(N+1)}{2}$ biases. The genetic algorithm itself has an additional two parameters, P_1 and P_2, for use as multipliers (along with $C_{\max}$) on the link and node biases, respectively. The cost matrix is then biased by both of these values using

$$C'_{ij} = C_{ij} + P_1 C_{\max} b_{ij} + P_2 C_{\max} (b_i + b_j)$$

The parameters P_1 and P_2 are fixed for a single genetic algorithm experiment. For the OCSTP, we believe that the b_{ij} biases are unimportant and so we ran our experiments

with $P_1 = 0$ and $P_2 = 1$. For our experiments we allowed the b_{ij} to take on values in the range $[0, 255]$.

We made several runs using this new representation and found it to be quite effective. With only a little experimentation we found a single set of genetic algorithm parameters consistently found excellent solutions that were as good or better than a good heuristic algorithm (see section 3.4). Because of this success and because we increased the number of parameters by two, thereby pushing the size of the search space up to 10^{12}, we chose not to invest the extensive machine time required to run the meta-GA to search for the best parameter set. The parameters that we chose were:

Population	100	σ *Scaling*	1.0
Crossover	0.6	P_1	0.0
Mutation	0.01	P_2	1.0
Gen Gap	1.0		

We now present some computational experience with this representation.

3.4. Results

Once we had established guidelines for the parameters best suited for our link and node biased (LNB) genetic algorithm for the OCSTP, we made several runs of the genetic algorithm for each of the five problems. Since genetic algorithms are randomized procedures, we averaged several runs for each problem to obtain our results. We evaluated the effectiveness of the LNB genetic algorithm in several ways. We first compared the results to a purely random search. We then compared the genetic algorithm's performance to that of a good, known heuristic algorithm. Finally, we changed the characteristics of the underlying network to strongly encourage a two-level, multiple-star, network in order to see how well the two techniques would adapt.

3.4.1. Random search comparison. We compared the distribution of solutions found by a purely random search to the distribution of the solutions to a 24-node problem found by the genetic algorithm. As shown in figure 3, the genetic algorithm using 10,000 trials consistently found solutions superior to a random search of one million solutions by 4 standard deviations.

3.4.2. Heuristic algorithm comparison. When we compare the costs of the trees found by the LNB genetic algorithm when using the parameters identified in

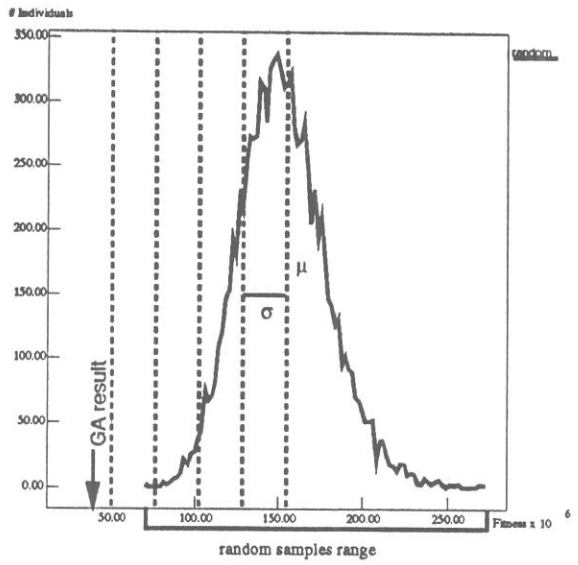

Figure 3: GA comparison with one million random samples for N=24.

section 3.3.2 to the trees found by the heuristic in table, we find that the genetic algorithm consistently found solutions as good as or better than those found by the heuristic.

N	*Heuristic*	*GA minimum*	*GA average*	*GA maximum*	*GA gain*
6	1,386,360	1,386,360	1,412,880	1,419,540	0%
12	7,134,530	6,857,020	6,857,020	6,857,020	3.9%
24	38,082,652	36,029,600	36,029,600	36,029,600	5.4%
47	153,584,714	142,513,390	143,430,000	144,447,260	5.9-7.1%
98	751,893,094	703,800,360	711,908,800	718,407,620	4.5-6.4%

Of course, halting a genetic algorithm after some arbitrary number of evaluations may not allow the genetic algorithm to complete its work. In our experiments, the LNB genetic algorithm encoding easily provided very good solutions within the 10,000 evaluations budget we used. While competing with the heuristic approach was not our purpose in this study, it proved very interesting to compare the resulting trees found by the two approaches (see figures 4 and 5).

Figure 4: Heuristic solution for $N = 98$.

3.4.3. Adaptation to modified problem. Part of our goal was to design a genetic algorithm that could be relied upon to produce very good solutions for the OCSTP even when faced with the kinds of changes to the parameters of the problem that might cause a heuristic to break down. When someone designs a new heuristic, a reliable yardstick is needed in order to evaluate the heuristic. Older heuristics might be employed for this purpose, but they too are subject to changes in the problem definition and it may not be known whether they are finding really good solutions or just the best ones known. A genetic algorithm that can adapt to changing problem parameters and still produce reliably good groups of solutions would be preferable.

When the problem parameters change, the heuristic must be reexamined to see if it will continue to produce good solutions. In our heuristic, the quite reasonable assumption "one or two stars are good" was one of the rules used. If we change the problem parameters so that link costs would encourage the use of several stars, this part of the heuristic was no longer able to produce good solutions. The genetic algorithm, which makes use of no problem-specific knowledge, adapted well to the problem and continued to produce good solutions. Results from running the heuristic

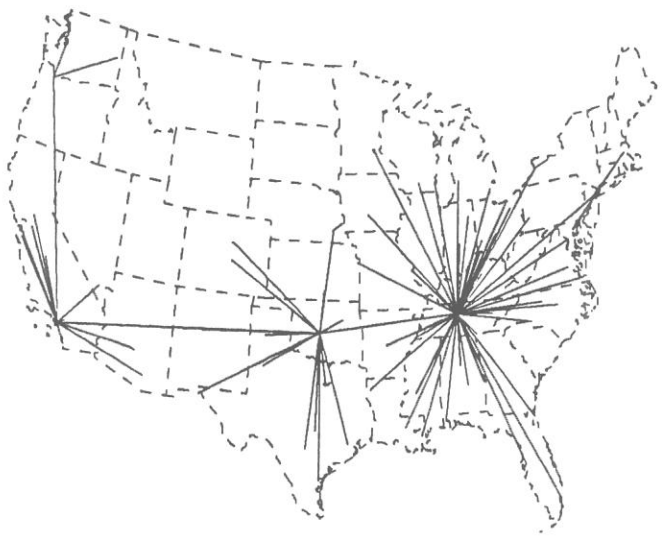

Figure 5: LNB genetic algorithm solution for $N = 98$.

and the genetic algorithm on the modified problem are shown below:

$$\begin{array}{ll} Heuristic & 3,210,004 \\ LNB\ GA & 2,367,504(optimum) \end{array}$$

The trees found by the LNB genetic algorithm and the heuristic for the modified problem with $N = 24$ are shown in figures 6 and 7, respectively. Due to the much simplified traffic patterns in the modified problem, we were able to determine that the LNB genetic algorithm was, in fact, finding the optimal solution.

4. Conclusions and Future Work

We have defined a representation which satisfies all but one of the criteria given in section 3.1 and which can be used effectively to represent trees within a GA. We are planning to run a meta-GA on the LNB genetic algorithm in order to see what the best genetic algorithm control parameters really are and to verify our intuition about the unimportance of the b_{ij} values to the OCSTP. However, we are also aware that because we are ignoring the b_{ij} values by setting $P_1 = 0$, this representation has an inherent bias toward star-based networks, since it tends to simultaneously favor all links from nodes with small bias. Even though this bias had no apparent effect on

Figure 6: LNB GA network for modified problem with $N = 24$.

the OCSTP problems we considered, we will be investigating how this representation fares when faced with an OCSTP problem in which severe degree constraints (i.e. only degrees of 1 or 2) are placed on the nodes.

Recent work by Orvosh and Davis [10] indicates that when an encoding allows invalid chromosomes to be produced by mating, it isn't always best to "repair" them. They suggest that this reparation process can result in the loss of important genetic material and, thus, reduce the efficiency of the search. Their proposed alternative is another genetic algorithm parameter which specifies the probability that an invalid chromosome is repaired. Their research has shown good success with $P_{repair} \approx 5\%$. We would like to investigate this new idea to see if its application would allow more effective use of one of the other, more fragile, encodings.

References.

[1] L. Davis. Job shop scheduling with genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 136–140, Carnegie-Mellon University, Pittsburgh, PA, July 24-26 1985. Lawrence Erlbaum Associates.

Figure 7: Heuristic network for modified problem with $N = 24$.

[2] K. A. DeJong. *Analysis of the Behavior of a class of genetic adaptive systems.* PhD thesis, Department of Computer and Communications Sciences, Univ. of Michigan, 1975.

[3] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, New York, NY, 1985.

[4] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA, 1989.

[5] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[6] J. H. Holland. Hierarchical descriptions of universal spaces and adaptive systems. Technical Report ORA Projects 01252 and 08226, University of Michigan, Department of Computer and Communication Sciences, Ann Arbor, MI, 1968.

[7] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, MI, 1975; reprinted by MIT Press, Canbridge, MA, 1992, 1992.

[8] T. C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[9] J. W. Moon. Various proofs of cayley's formula for counting trees. In F. Harary, editor, *A Seminar on Graph Theory*, pages 70–78. Holt, Rinehart, and Winston, New York, NY, 1967.

[10] D. Orvosh and L. Davis. Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 650, University of Illinois at Urbana-Champaign, July 17-21 1993. Morgan Kaufmann.

[11] N. N. Schraudolph and J. J. Grefenstette. A user's guide to gaucsd 1.4. Technical Report CS92-249, UC San Diego, CSE Department, UC San Diego, La Jolla, CA 92093-0114, 1992.