Representing Trees in Genetic Algorithms

Charles C. Palmer, Aaron Kershenbaum
IBM T.J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
cpalmer@watson.ibm.com, kersh@watson.ibm.com

***Abstract*— We consider the problem of representing trees (undirected, cycle-free graphs) in Genetic Algorithms. This problem arises, among other places, in the solution of network design problems. After comparing several commonly used representations based on their usefulness in genetic algorithms, we describe a new representation and show it to be superior in almost all respects to the others. In particular, we show that our representation covers the entire space of solutions, produces only viable offspring, and possesses locality, all necessary features for the effective use of a genetic algorithm. We also show that the representation will reliably produce very good, if not optimal, solutions even when the problem definition is changed.**

I. Introduction

In this paper, we consider the problem of representing trees in genetic algorithms. A tree is an undirected graph which contains no closed cycles. There are many optimization problems which can be phrased in terms of finding the optimal tree within a given graph or network. Sometimes any tree is permitted, as in the case of a minimum spanning tree or shortest path tree [2]. In other cases, the tree might be constrained. For example, the degrees of some or all of the nodes in the tree might be limited. While some problems using trees are easy, for many others (e.g., the Steiner Tree Problem , Capacitated Minimum Spanning Tree Problem , and Travelling Salesperson Problem [2][4][1]) there is no practical means of obtaining a guaranteed optimal solution in a reasonable amount of time. In such cases, a genetic algorithm (GA) may be the best means available to obtain good solutions to the problem.

In particular, we were interested in finding solutions to the Optimal Communications Spanning Tree Problem (OCSTP) [3]. The goal of this problem is to find a tree (cycle-free graph) of minimum total cost satisfying a given requirement set. Thus we seek the tree, T, which minimizes

$$X(T) = \sum_{i,j} A_{ij}(T) \times C_{ij} \qquad (1)$$

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

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 such as genetic algorithms.

II. Criteria for Selecting Tree Representations in Genetic Algorithms

In order for a GA to function most effectively, the representation, or encoding, of trees must possess 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 more effectively by having the encoding truly represent the tree.

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.

III. 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 [1] 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.

A. 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, $\ldots K$, 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 $2^{-\left[N\left(\frac{N}{2}-\log 2(N)\right)\right]}$.

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 $O\left(N^2\right)$ 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 $O\left(N\right)$ 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; changing 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.

B. *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.

C. *Prüfer Numbers*

A third possible encoding is the Prüfer number [5] 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, below. 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 $P(T) = 3311$.

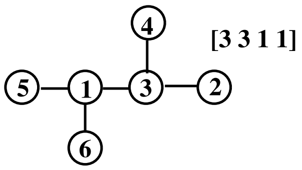

Figure 1: A Tree and its Prüfer number.

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.

IV. A New Representation for Trees

We set off to design a new representation for trees that would satisfy the criteria given in section II. 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 performed poorly [7]. 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 began anew by noting that 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

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

The tree that the chromosome represents is then found by applying Prim's algorithm [1] 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 determine 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 we want to encode the tree having

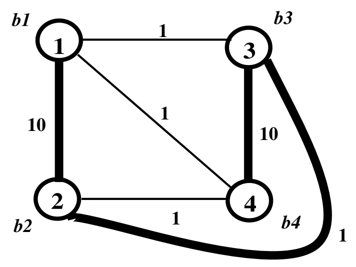

Figure 2: Example of a tree not representable using only node biases.

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, we may obtain several inequalities such as

$$C'_{12} \leq C'_{13} \quad (2)$$

$$C'_{12} \leq C'_{23} \quad (3)$$

since $(1,2)(2,3)(1,3)$ form a cycle. Similarly, if $(3,4)$ is to be part of the MST

$$C'_{34} \leq C'_{23} \quad (4)$$

$$C'_{34} \leq C'_{24} \quad (5)$$

But, these inequalities contain contradictory pairs. For example,

$$\begin{aligned} (2) \Rightarrow \quad & C'_{12} \leq C'_{13} \\ & b_2 + 9 \leq b_3 \\ (5) \Rightarrow \quad & C'_{34} \leq C'_{24} \\ & b_3 + 9 \leq b_2 \end{aligned}$$

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. While a tree like the chain above is clearly not useful as a solution to the OCSTP, 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

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

This version of the representation can encode any tree, T, given suitable values of the b_i. This is easily seen by observing that we can set $b_i = 0$, for all i, and

$$b_{ij} = \begin{array}{l} 0 \; for \; (i,j) \in T \\ M \; otherwise \end{array}$$

where M is larger than the maximum value of C_{ij}. 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_i, b_j, and 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. 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		

V. Conclusions and Summary

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.

A. Random search comparison

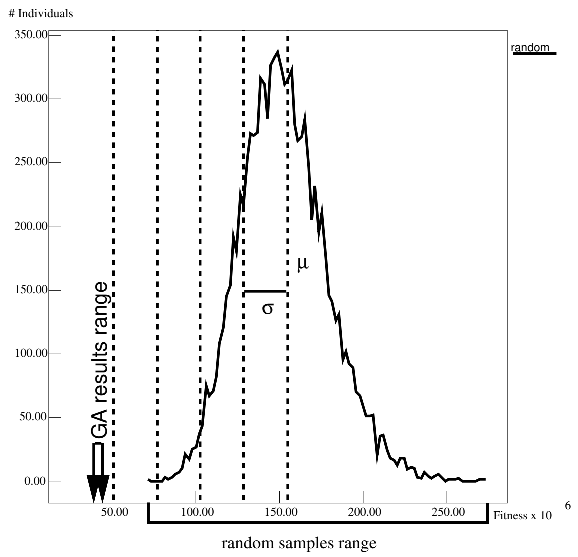

Figure 3: One million random samples for N=24.

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 more than 4 standard deviations.

B. Heuristic algorithm comparison

Since optimal solutions to the OCSTP are not generally known, we compared our genetic algorithm with a good heuristic that has been used for the OCSTP in the past [7]. This heuristic first searches for the best tree that is a star on one node, then it searches for the best tree that is a connected pair of stars, and finally, starting with a MST, it tries to reduce the number of interior nodes by redirecting links to leaf nodes until it can improve no more. The heuristic returns the best tree found during these three steps. When we compare the costs of the trees found by the LNB genetic algorithm when using the parameters identified in section IV to the trees found by the heuristic, we find that the genetic algorithm consistently found solutions as good as or better than those found by the heuristic:

N	GA	$Heuristic$	GA $improvement$
6	$1,386,360$	$1,386,360$	0%
12	$6,857,020$	$7,134,530$	3.9%
24	$36,029,600$	$38,082,652$	5.4%
47	$143,949,400$	$153,584,714$	6.3%
98	$711,287,300$	$751,893,094$	5.4%

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.

C. Adaptation to modified problem

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 original problem, the requirements between nodes was inversely proportional to their distance apart. While this is a reasonable model of some networks, we decided to change it to model another kind of network based on distribution centers. In the new problem, all of the requirements were regionalized into groups and were exclusively between leaf nodes and their "center", with a token requirement between the centers.

For this new problem, the heuristic was unable to produce good solutions. The LNB genetic algorithm, which makes use of no problem-specific knowledge, adapted well to the problem and continued to produce good solutions. The results from running the heuristic and the genetic algorithm on the modified problem are shown below:

Heuristic	$3,210,004$
LNB GA	$2,367,504(optimum)$

Due to the much simplified traffic patterns in the modified problem, we were able to prove that the LNB genetic algorithm was, in fact, finding the optimal solution [7].

VI. Conclusions and Future Work

We believe that we have defined a representation which satisfies most of the criteria given in section II. 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. Even though this bias had no effect on 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 [6] 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] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, New York, NY, 1985.

[2] M. Gondran and M. Minoux. *Graphs and Algorithms.* John Wiley, New York, NY, 1984.

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

[4] A. Kershenbaum. *Telecommunications Network Design Algorithms.* McGraw-Hill, New York, NY, 1992.

[5] 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.

[6] 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.

[7] C. C. Palmer and A. Kershenbaum. Two algorithms for finding optimal communications spanning trees. Technical Report (report number unassigned), IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY, 1993.