Proceedings of The Sixth International Conference on Genetic Algorithms (ICGA-95), Pittsburgh, PA., July, 1995, Larry Eshelman, Editor, Morgan Kaufmann Publisher, pp. 470-477.

Determinant Factorization: A New Encoding Scheme for Spanning Trees Applied to the Probabilistic Minimum Spanning Tree Problem

Faris N. Abuali
Math and Computer Sciences
The University of Tulsa
abuali@euler.mcs.utulsa.edu

Roger L. Wainwright
Math and Computer Sciences
The University of Tulsa
rogerw@penguin.mcs.utulsa.edu

Dale A. Schoenefeld
Math and Computer Sciences
The University of Tulsa
dschoen@euler.mcs.utulsa.edu

Abstract

This paper describes a new encoding scheme for the representation of spanning trees. This new encoding scheme is based on the factorization of the determinant of the in-degree matrix of the original graph. Each factor represents a spanning tree if the determinant corresponding to that factor is equal to one. Our new determinant encoding will be compared to the Prüfer encoding, and to the node and link biased encoding by solving an NP-complete variation of the minimum spanning tree problem, known as the Probabilistic Minimum Spanning Tree Problem. Given a connected graph *G(V,E)*, a cost function c:$E \longrightarrow \Re^+$, and a probability function P:$2^V \longrightarrow [0,1]$, the problem is to find an *a priori* spanning tree of minimum expected length. Our results show a significant improvement in using the new determinant encoding and the node and link biased encoding compared to Prüfer's encoding. We also show empirically that our new determinant encoding scheme is as good as the node and link biased encoding. Our new determinant encoding works very well for restricted spanning trees, and for incomplete graphs.

1 INTRODUCTION

The classical minimum spanning tree (MST) is used to help solve many combinatorial optimization problems. The MST problem has important applications in transportation, communication network design, and distribution systems.

Recently Bertsimas (1990) defined the Probabilistic Minimum Spanning Tree (PMST), which is a variation of the minimum spanning tree where each vertex is present with some given probability. The PMST model is more realistic and representative of many combinatorial optimization problems than the minimum spanning tree, especially if the MST is NOT the "best" solution for all instances of the problem. When the vertex probability is one for each vertex, the PMST problem reduces to the MST problem. Many applications are natural for the PMST such as VLSI design, communication network design, and organizational structures design.

In Section 2 a formal description of the PMST problem is presented. In Section 3 we discuss Prüfer's encoding with some examples. Section 4 gives a review of the Link and Node Biased (LNB) encoding scheme. Section 5 gives an introduction to the determinant factorization and defines our new determinant encoding scheme. Section 6 describes the genetic algorithm (GA) test cases and the results of our experiments.

2 THE PMST PROBLEM

Bertsimas has formally defined the Probabilistic Minimum Spanning Tree (PMST) problem as follows (Bertsimas (1990)): Given a connected undirected graph *G=(V,E)* , not necessarily complete, a cost function c:$E \longrightarrow \Re^+$, and a probability function P:$2^V \longrightarrow [0,1]$, the objective is to find a spanning tree, say T, that minimizes the expected active cost, $E[L_T]$, where

$$E[L_T] = \sum_{S \subseteq V} p(S) L_T(S).$$

The above summation is taken over all subsets of V, *T(S)* is the minimum subtree of T that is required to interconnect all the nodes in S, and $L_T(S)$ is the total cost of the edges in *T(S)*.

If one assumes that node i is active with probability p_i and that the nodes are independent, Bertsimas shows that the expected active cost, $E[L_T]$, for a given spanning tree is given by the expression

$$E[L_T] = \sum_{e \in T} c(e) \left\{ 1 - \prod_{i \in K_e} (1 - p_i) \right\} \times$$

$$\left\{1-\prod_{i\in V-K_e}(1-p_i)\right\}.$$

The above summation is over all edges in the tree, T. The removal of an edge, e, splits T into two subtrees. The set of vertices in one subtree is denoted by K_e and the set of vertices in the other subtree is denoted by $V-K_e$. The above summation computing $E[L_T]$ is $O(n^2)$ where n is the number of vertices. Bertsimas also shows how to compute $E[L_T]$ in $O(n)$ time.

3 PRÜFER'S ENCODING

One of the classical theorems in graphical enumeration is Cayley's theorem (Cayley (1889)) that there are $n^{(n-2)}$ distinct labeled trees on a complete graph with n vertices. Prüfer (Prüfer (1918)) provides a constructive proof of Cayley's theorem by establishing a one-to-one correspondence between such trees and the set of all strings of $n-2$ integers, where each integer is between 1 and n inclusive.

The key to Prüfer's code is the observation that for any tree there are always at least two vertices of degree one (Skiena (1990)). Hence, in a labeled tree T, the vertex v incident to the lowest labeled vertex of degree one is uniquely determined, and v becomes the first symbol in the string. After deleting this edge we have a tree with $n-1$ vertices. Repeating this operation until one edge is left produces $n-2$ integers between 1 and n inclusive (Skiena (1990)).

To reconstruct a tree T from Prüfer's code, we note that any vertex will appear in the code exactly one time less than the degree of the vertex in T. Leaf nodes, which have degree one, will not appear in the code. Thus, it is possible to compute the degree of all the vertices of T, and identify the lowest labeled degree-one vertex in the tree, say u. Since the first symbol in the code is the vertex incident to u, the first edge is easily determined, and by repeating this operation, the remaining edges can be determined. An example is shown below illustrating Prüfer's encoding.

The chromosome corresponding to the spanning tree of a complete graph on nine vertices represented in Figure 1 is (2 7 3 1 7 2 1). The construction of the chromosome encoding of Figure 1 is described as follows. Locate the node of degree one having the smallest label. In this case, it is node 4. Since node 2 is the node (the only node) in the tree that is adjacent to node 4, we assign 2 to the first allele in the corresponding chromosome. We continue the process on the subtree resulting from removing node 4 and the edge from node 2 to node 4. We summarize this first step and the successive steps of the construction:

2 becomes first allele, removing node 4 and edge (2,4),
7 becomes next allele, removing node 5 and edge (5,7), and
3 becomes next allele, removing node 6 and edge (3,6),
1 becomes next allele, removing node 3 and edge (1,3),
7 becomes next allele, removing node 8 and edge (7,8),
2 becomes next allele, removing node 7 and edge (2,7), and
1 becomes next allele, removing node 2 and edge (1,2).

The tree that remains consists of nodes 1 and 9 and the edge (1,9). The algorithm stops when the remaining tree has two nodes and one edge remaining.

Conversely, since the procedure is not immediately obvious, we also illustrate the strategy to construct a spanning tree in a complete graph on nine nodes from a given chromosome (Skiena (1990)). The key observation is that the degree of each labeled node is one more than the cardinality of the node number in the chromosome. For the chromosome (2 7 3 1 7 2 1), the degree of node 1, 2, 3, 4, 5, 6, 7, 8 and 9 is, respectively, 3, 3, 2, 1, 1, 1, 3, 1 and 1.

The smallest labeled node of degree one is node 4. Since 2 is the first allele in the chromosome, node 4 must be adjacent to node 2. After inserting an edge from node 4 to node 2, we decrement the remaining degree of node 2 from 3 to 2, and decrement the remaining degree of node 4 from 1 to 0. The process then repeats.

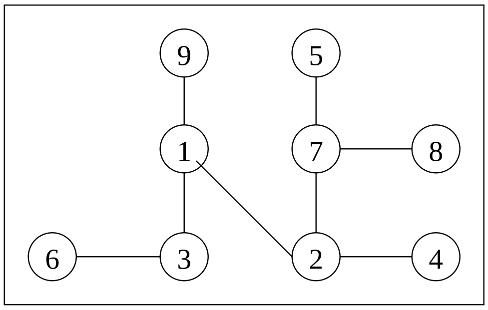


Figure 1: An Example Tree With Prüfer Encoding Chromosome (2 7 3 1 7 2 1).

4 THE LINK AND NODE BIASED ENCODING

The link and node biased (LNB) encoding was developed by Charles Palmer in his Ph.D. dissertation (Palmer (1994)). His idea is described as follows: Given a complete graph $G(V,E)$, with n vertices, and a cost function $C: V\times V \longrightarrow \Re^+$, then the LNB encoding for a spanning tree is a vector consisting of link biases and node biases. Each *link bias* and each *node bias* is an integer in the range 0..255. The spanning tree corresponding to the LNB code is found by running a minimum spanning tree algorithm on the modified cost function C':

$$C'_{i,j} = C_{i,j} + P_1\, b_{i,j}\, C_{max} + P_2\,(b_i + b_j)\, C_{max}.$$

C_{max} is the maximum link cost in the graph, $b_{i,j}$ is the link bias associated with the edge from node i to node j, and b_i is the node bias associated with node i. The node and link biases are normalized to map to the range $[0,1]$ before using them in the equation above. P_1 and P_2 are control parameters. Palmer used $P_1 = 0$ and $P_2 = 1$ in his experiments.

Notice that one can generalize the Link and Node Biased Encoding. Consider the general concept utilized by the LNB encoding. There are four components as follows:

1. Start with some desired network topology. For example, a spanning tree.
2. Given that there is an algorithm to find the desired topology as a function of some parameter, for example a node to node cost, we change or bias the value of that parameter appropriately.
3. Run some desired network generation algorithm, for example a minimum spanning tree (MST), using the biased parameter.
4. Finally, run the objective function on the result of the previous step.

5 DETERMINANT ENCODING

5.1 THE NUMBER OF SPANNING TREES

Even (1979) describes the following technique for finding the number of spanning trees in a directed graph (developed by Tutte (1948)):

The in-degree matrix D of a digraph *G(V,E)* is defined as follows:

$$D(i,j) = \begin{cases} d_{in(i)} & \text{if } i = j, \\ -k & \text{if } i \neq j \end{cases}$$

where $d_{in(i)}$ is the number of edges coming into node i, and k is the number of edges in G from i to j.

Lemma 1 (Even (1979)) *A finite digraph G(V,E), with no self-loops is a directed tree with root r if and only if its in-degree matrix* D *has the following two properties:*

1. *The root has no edges coming in, and all other nodes have one edge coming in, formally,*

$$D(i,i) = \begin{cases} 0 & \text{if } i = r, \\ 1 & \text{if } i \neq r. \end{cases}$$

2. *The minor, resulting from erasing the* rth *row and column from* D *and computing the determinant, is 1.*

Theorem 1 (Even (1979)) *The number of directed spanning trees with root r of a digraph with no self-loops is given by the minor of its in-degree matrix which results from the removal of the* rth *row and column.*

The PMST problem is defined on an undirected graph *G(V,E)*, and any solution to the PMST problem is an undirected tree T. To see the relationship between directed and undirected spanning trees, consider the digraph $G'(V,E')$ defined as follows: For every edge $u \overset{e}{-} v$ in G define the two edges $u \overset{e'}{\rightarrow} v$ and $v \overset{e''}{\rightarrow} u$ in G'. Regardless of the choice of $r \in V$, there is a one-to-one correspondence between the set of spanning trees of G and the set of directed spanning trees of $G'(V,E')$ with root r: Let T be a spanning tree of G. If the edge $u \overset{e}{-} v$ is in T and if u is closer than v to r in T then pick e' for T'; if v is closer, pick e''. Also, given T', it is easy to find the corresponding T by simply ignoring the directions; i.e., the existence of either e' or e'' in T' implies that e is in T (Even (1979)). Defining the *degree* matrix of G to be the in-degree matrix of G', we can state:

Theorem 2 (Even (1979)) *The number of spanning trees of an undirected graph with no self-loops is equal to any of the minors of its degree matrix which results from the erasure of a row and a corresponding column.*

An example is shown below illustrating the determinant factorization.

Given the complete digraph of 3 vertices the in-degree matrix D is:

$$D = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}.$$

If we remove the first row and column (assuming vertex 1 to be the root), and compute the determinant of the resulting matrix (i.e the number of spanning trees, T) we get:

$$T = \begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix} = 4 - 1 = 3.$$

To find these spanning trees it is necessary to decompose the above determinant into columns which represent an edge in every column. First, the 2×2 determinant can be written as

$$\begin{vmatrix} 1 & -1 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix}.$$

In this instance we have reinserted the first row of D except its first entry, which we make equal to 1. All other entries in the first column are changed to zero. Next, using linearity of the determinant function and simple observation, we can decompose each column (except the first) into columns which consist of a single +1 and a single -1, as shown in Figure 2.

$$\begin{vmatrix} 1 & -1 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix} = \begin{vmatrix} 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{vmatrix}$$

Figure 2: The Determinant Factors For A Complete Graph With 3 Vertices .

Each of the four determinants on the right side of the equal sign, shown in Figure 2, corresponds to a selection of n-1 edges of the original graph. Figure 3 shows, in respective order, the four graphs corresponding to the factors (left to right) in Figure 2.

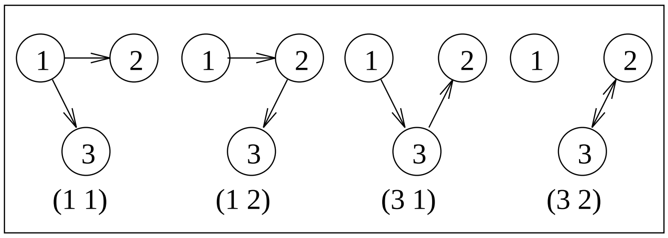
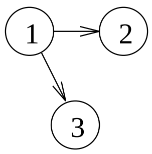


Figure 3: The 4 Graphs Corresponding To The Determinant Factors For The 3 Node Complete Graph. The 3 Valid Spanning Trees Are Denoted By Boldface.

For example in Figure 2 and Figure 3 the determinant

$$\begin{vmatrix} 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

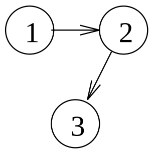
corresponds to the tree



The directed edge from node 1 to node 2 results from the -1 entry in row 1 and column 2. The directed edge from node 1 to node 3 results from the -1 entry in row 1 and column 3. Similarly, the determinant

$$\begin{vmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{vmatrix}$$

corresponds to the tree



The determinant

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{vmatrix} = 0$$

does NOT represent a valid tree. Any factor that evaluates to zero is NOT a valid tree. The meaning of the 2-digit code shown in parenthesis in Figure 3 below each graph will be explained in the next section.

5.2 THE GA ENCODING SCHEME

Several researchers have investigated the benefits of solving combinatorial optimization problems using genetic algorithms (Abuali, Schoenefeld, and Wainwright (1993, 1994a, 1994b), Blanton and Wainwright (1993), Corcoran and Wainwright (1992), De Jong and Spears (1989), and Whitley, Starkweather, and Fuquay (1989)). Davis (1987, 1991), Goldberg (1989) and Rawlins (1991) provide an excellent in depth study of genetic algorithms. It is assumed that the reader is familiar with the fundamentals of genetic algorithms(GA). The GA package used in this research is LibGA (Corcoran and Wainwright (1993)).

Define the *edge-range vector* R of a digraph *G(V,E)* as follows:

$$R(j) = \{i \mid D(i,j) = -1\} \text{ for } j = 1 \text{ to } n$$

where D is the in-degree matrix. Furthermore, define a *determinant code*, DC to be a string of $n-1$ integers, where n is the number of vertices in the original graph, and the $j-1$ position in the string is selected from *R(j)* for $j = 2$ to n. For example, the set of all *determinant codes* for a 3 vertex graph is given by: $\{(x_2, x_3) \mid x_j \in R(j)\ \forall j = 2, 3\}$. If DC is a *determinant code*, the $j-1$ position in DC represents the row index of an entry of -1 in the j-th column of the in-degree matrix. A k in the $j-1$ position of DC corresponds to an edge from vertex k to j. Each spanning tree corresponds to a *determinant code*. However, there are "infeasible" *determinant codes* that do not correspond to spanning trees. This is the *determinant encoding* scheme used for our GA.

The *edge-range vector*, R, for the complete graph on 3 nodes is

$$(\{2,3\}, \{1,3\}, \{1,2\}).$$

Since $R(2) = \{1,3\}$ and $R(3) = \{1,2\}$, the set of all *determinant codes* is

$$\{(1,1),(1,2),(3,1),(3,2)\}.$$

The subgraphs corresponding to the determinant codes are labeled in Figure 3.

As another example, the 4 node complete graph has the following in-degree matrix,

$$D = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}.$$

The *edge-range vector*, R, is

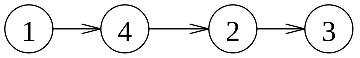
$$(\{2,3,4\},\{1,3,4\},\{1,2,4\},\{1,2,3\}).$$

and the set of *determinant codes* has twenty seven elements determined by $R(2)$, $R(3)$, and $R(4)$.

Assuming that vertex one is the root and selecting

$$\begin{vmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{vmatrix}$$

as one of the determinant factors, the corresponding *determinant code* DC is (4 2 1). This follows since -1 is in row 4 of column 2, -1 is in row 2 of column 3, and -1 is in row 1 of column 4. The corresponding edges are $4 \to 2$, $2 \to 3$, $1 \to 4$. The resulting spanning tree is



.

5.3 REPAIRING DETERMINANT CODES

The determinant encoding scheme sometimes produces graphs which are NOT spanning trees. The determinant encoding scheme has the following properties. If we ensure that the root is included in the determinant code, then any component in the graph resulting from the code will have two or more vertices. Furthermore, one of the components will be a tree component. Each of the other components will contain only one cycle. This is true because any given vertex has an in-degree of one, and therefore any component will only have one cycle at most.

Our algorithm for repairing determinant codes is described as follows:

1. Scan the determinant code, DC, to ensure that the root is in the code. If the root is not already in the code, replace a randomly selected allele with the root.
2. Identify the set of components, C, in the graph resulting from the determinant code in step 1.
3. If the number of connected components is equal to one, then the determinant code represents a tree, and no repair is required. Otherwise perform the repair described in step 4.
4. Let NC be the number of components in the graph represented by the determinant code. Let T be the component that is a tree. Let $C_1, C_2, \ldots, C_{NC-1}$ be the various other components. Let T be the component that is a tree.

   While the number of connected components NC is greater than one, repeat the following steps:

   a) Select randomly a vertex, i, from T to become the parent.
   b) Select randomly a component, C_k.
   c) Select randomly a vertex, j, from the vertices forming the cycle in component C_k to become the child of parent i.
   d) Let $DC(j-1) = i$.
   e) Let $T = T \bigcup V(C_k)$, where $V(C_k)$ returns the vertices in C_k.
   f) Let $NC = NC - 1$.

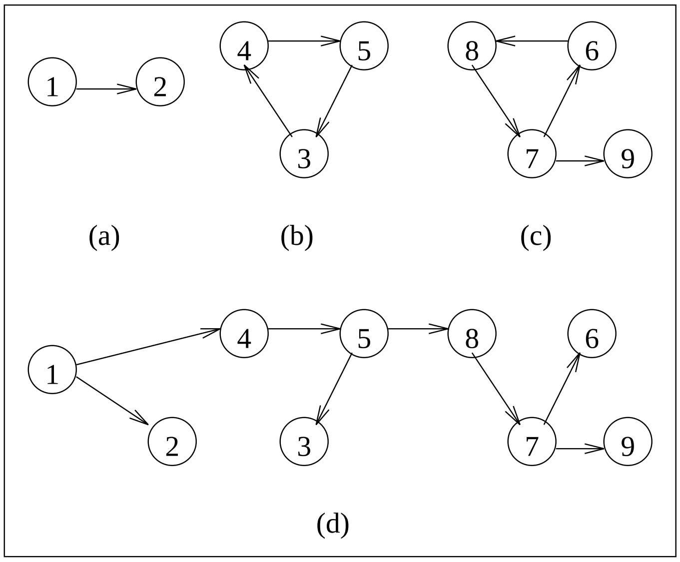


Figure 4: The Components Corresponding To The Determinant Code (1 5 3 4 7 8 6 7). (a) Shows The Tree Component T, (b) Shows The Cycle Component C_1, (c) Shows The Second Cycle Component C_2, (d) Show The Spanning Tree Resulting From The Determinant Repair Algorithm.

The determinant code (1 5 3 4 7 8 6 7) for a complete graph on 9 nodes corresponds to the subgraph represented by Figure 4(a), Figure 4(b), and Figure 4(c). The repair described in step 4 produces the spanning tree in Figure 4(d) as follows. Select node 1 in the tree component, T, Figure 4(a). Select component C_1, Figure 4(b). Select node 4 in component

C_1. Let $DC(3) = 1$, which connects node 1 to node 4, and also disconnects the edge from node 3 to node 4. Now there are two components left, a tree which combined Figure 4(a) and 4(b), and component C_2, Figure 4(c). We repeat step 4 of the algorithm one more time. Select node 5 from the tree; select node 8 from C_2. Now an edge from node 5 to node 8 is generated when $DC(7) = 5$. This has the side effect of removing the edge from node 6 to node 8. The resulting determinant code for Figure 4(d) is (1 5 1 4 7 8 5 7).

6 GENETIC ALGORITHM TEST CASES AND RESULTS

6.1 TEST CASES

Ten data sets with 20, 30, and 40 nodes each were constructed (producing a total of 30 cases). In each case a complete graph was generated. The nodes were randomly placed on a 100×100 grid. The cost of each edge is the Euclidean distance. The probability associated with each node in all cases was fixed at 0.1. Each data set used a different random number seed to generate random coordinate positions for the nodes. For each of the test cases the expected active cost of the classical MST was compared to the expected active cost of various spanning trees evolved by applying genetic algorithm strategies to three different encodings of spanning trees, the Prüfer encoding, the Link and Node Biased encoding, and our new Determinant Encoding.

6.2 GA PARAMETERS

For the Prüfer encoding scheme we used the crossover operator *Alter Allele* and the mutation operator *Alter Allele*. From our previous research work, we have determined these two crossover and mutation operators generally produce the best results when used with the Prüfer encoding (Abuali, Schoenefeld, and Wainwright (1994a)). For the Link and Node Biased encoding we used the crossover operator *uniform* and the mutation operator *swap*. The mutation rate was 0.001. For the determinant encoding scheme we used the crossover operator *uniform* and the mutation operator *Alter Allele*. The mutation rate for both the Prüfer encoding and the determinant encoding was set dynamically, based on the population's fitness variance (as the fitness variance decreased the mutation rate increased.)

6.3 RESULTS

Table 1 shows the results of the PMST problem for the 20 node data sets. The greedy algorithm used for constructing the spanning trees was the classical MST algorithm. The expected active cost of the MST was then computed. The expected active cost for the PMST problem using the greedy strategy, the Prüfer encoding, the determinant encoding without repair and with repair, and the Link and Node Biased encoding are shown for each of the 10 random test cases. The GA using the Prüfer encoding produces the same answer as the determinant encoding (no repair) and the Link and Node Biased encoding in only one test case (number VII). The determinant encoding without repair ties with the Link and Node Biased encoding in three out of the ten test cases. The determinant encoding with repair tied with the LNB encoding in 4 of the 10 test cases. For test case III the determinant encoding produced better result than the LNB encoding. The "*" in Table 1, Table 2, and Table 3 indicates best performers.

The results for the 30 node, and 40 node data sets are shown in Table 2 and Table 3, respectively. Table 2 shows that the determinant encoding producing better results than the LNB encoding for 4 of the 10 test cases. This demonstrates that our determinant encoding scheme is competitive with other well known schemes for representing spanning trees. Table 3 shows that determinant encoding (No Repair) produced a better result than the LNB encoding for only one of the test cases.

7 CONCLUSIONS

This paper describes a new encoding scheme of spanning trees. This new method, the determinant encoding, explores the fact that all undirected spanning trees can be derived from all directed trees rooted at node i (where i can be any node) by changing the directed edges to undirected edges. To the authors' knowledge this scheme has not been presented before. We compared our determinant encoding to the Link and Node Biased encoding developed recently by Palmer by applying these encodings to the Probabilistic Minimum Spanning Tree (PMST) problem. Our results show that our encoding is competitive with the LNB encoding, and in some test cases produced better results. The determinant encoding scheme also works well on other types of graphs such as incomplete graphs, and restricted spanning trees. The determinant encoding scheme represents an alternative encoding scheme for representing trees in genetic algorithms.

Acknowledgements

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

References

F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright (1993). The design of a multipoint line topology for a communication network using genetic algorithms. In John Y. Cheung (ed.), *Proceedings of the Seventh Oklahoma Symposium on Artificial Intelligence*, 101-110, Norman, Oklahoma.

F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright (1994a). Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In Ed Deaton, K. M. George, Hal Berghel, and George Hedrick, editors, *Proceedings of the 1994 ACM/SIGAPP Symposium on Applied Computing*, 242-246, New York: ACM Press.

F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright (1994b). Terminal assignment in a communications network using genetic algorithms. In Dawn Cizmar (ed.), *Proceedings of the Twenty Second Annual ACM Computer Science Conference*, 74–81, New York: ACM Press.

D. J. Bertsimas (1990). The probabilistic minimum spanning tree problem. *Networks*, 20:245-275.

J. L. Blanton and R. L. Wainwright (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In Stephanie Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, 452-459. Urbana-Champaign, Illinois: Morgan Kaufmann.

A. Cayley (1889). A theorem on trees. *Quarterly Journal of Mathematics*, 23:376-378.

A. L. Corcoran and R. L. Wainwright (1992). A genetic algorithm for packing in three dimensions. In Hal Berghel, Ed Deaton, George Hedrick, David Roach, and Roger Wainwright, editors, *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, 1021-1030. New York: ACM Press.

A. L. Corcoran and R. L. Wainwright (1993). LibGA: A user-friendly workbench for order-based genetic algorithm research. In Ed Deaton, K. M. George, Hal Berghel, and George Hedrick, editors, *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, 111-118. New York: ACM Press.

L. Davis (1987), editor. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann.

L. Davis (1991), editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York.

Kenneth A. De Jong and W. M. Spears (1989). Using genetic algorithms to solve np-complete problems. In Schaffer (1989), 124-132.

S. Even (1979). *Graph Algorithms.* Computer Science Press.

David E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts.

Charles Campbell Palmer (1994). *An Approach to a Problem in Network Design Using Genetic Algorithms.* PhD thesis, Polytechnic University.

H. Prüfer (1918). Neuer beweis eines satzes über permutationen. *Arch. Math. Phys.*, 27:742-744.

G. Rawlins (1991), editor. *Foundations of Genetic Algorithms.* Morgan Kaufmann.

J. David Schaffer (1989), editor. *Proceedings of the Third International Conference on Genetic Algorithms*, Arlington, Virginia: Morgan Kaufmann.

S. Skiena (1990). *Implementing Discrete Mathematics, Combinatorics and Graph Theory with Mathematica.* Addison Wesley.

W. T. Tutte (1948). The dissection of equilateral triangles into equilateral triangles. *Proc. Cambridge Phil. Soc.*, 44:463-482.

Darrell Whitley, T. Starkweather, and D. Fuquay (1989). Scheduling problems and traveling salesman: The genetic edge recombination operator. In Schaffer (1989).

Table 1: The Expected Active Cost For The PMST Problem, For 20 Nodes (Node Probability Is 0.1.)

Test Cases	Algorithms				
	Greedy	Prufer(GA)	Determinant(GA)		LNB(GA)
			No Repair	With Repair	
I	54.09	52.50	52.66	50.26*	50.26*
II	67.61	61.11	59.86	59.86	58.08*
III	62.57	56.97	56.11	55.58*	55.69
IV	87.09	69.53	63.52	64.75	62.44*
V	66.94	60.96	58.26*	58.26*	58.26*
VI	57.78	53.05	52.25*	52.25*	52.25*
VII	82.95	62.47*	62.47*	62.89	62.47*
VIII	69.38	64.57	59.15	57.92*	57.92*
IX	60.58	58.10	55.51	55.20	54.46*
X	65.77	60.96	59.45	59.45	59.32*

Table 2: The Expected Active Cost For The PMST Problem, For 30 Nodes (Node Probability Is 0.1.)

Test Cases	Algorithms				
	Greedy	Prufer(GA)	Determinant(GA)		LNB(GA)
			No Repair	With Repair	
I	94.29	95.77	87.22*	88.1	87.34
II	110.56	109.17	95.92*	97.34	95.98
III	109.29	100.69	96.75	96.11	94.23*
IV	122.97	99.95	90.66	90.83	90.23*
V	108.58	95.76	90.01*	90.01	90.11
VI	92.82	99.44	85.14	83.81	83.02*
VII	115.58	107.78	104.93	99.55	98.83*
VIII	100.81	99.21	95.21	92.67	92.41*
IX	106.28	103.26	109.88	93.25	91.26*
X	114.95	120.37	105.61	101.24*	101.51

Table 3: The Expected Active Cost For The PMST Problem, For 40 Nodes (Node Probability Is 0.1.)

Test Cases	Algorithms			
	Greedy	Prufer(GA)	Determinant(GA)	LNB(GA)
I	141.64	160.10	128.73	116.70*
II	166.40	150.22	138.84	136.86*
III	156.18	155.11	128.52*	129.32
IV	148.52	135.59	121.74	119.14*
V	159.45	159.45	134.12	130.84*
VI	128.33	136.20	117.36	116.72*
VII	164.41	181.17	135.73	130.76*
VIII	137.79	161.26	128.79	126.74*
IX	165.17	146.30	136.67	126.76*
X	131.09	143.28	126.45	124.51*