

## MINIMAL EDGE-ORDERED SPANNING TREES USING A SELF-ADAPTING GENETIC ALGORITHM WITH MULTIPLE GENOMIC REPRESENTATIONS

Richard Harvey  
Michael L. Gargano, [mgargano@pace.edu](mailto:mgargano@pace.edu)

Pace University Seidenberg School of Computer Science and  
Information Systems, Computer Science Dept., NYC, NY 10038

### ABSTRACT

We consider the problem of finding minimal edge-ordered spanning trees where the edge costs are not fixed, but are time dependent[7]. We propose using multiple genomic redundant representations in a self-adapting genetic algorithm (GA) employing various codes with different locality properties. These encoding schemes (e.g., permutation codes and prufer-like codes) either insure feasibility or require little repair after performing the operations of crossover and mutation and also ensure the feasibility of the initial randomly generated population (i.e., generation 0). The GAs applied in solving this NP hard problem employ non-locality or locality representations when appropriate (i.e., the GA adapts to its current search needs) which makes the GAs more efficient [15].

**Keywords:** minimal spanning tree, edge-ordered, self-adapting genetic algorithm, permutation codes, prufer-like codes

### *Introduction to the Problem*

We consider the problem of finding minimal edge-ordered spanning trees where the edge costs are not fixed, but are time dependent[7,19]. The cost of an edge depends on the time that it is constructed. Since finding such trees is an NP hard[7], we will solve it using genetic algorithmic methods. In particular, we propose using multiple genomic redundant representations in a self-adapting

genetic algorithm (GA) employing various codes with different locality properties. These GAs adapt to current search needs which makes the GAs more efficient in searching for very good albeit not necessarily optimal solutions.

### ***Mathematical Model***

Let  $G = (V, E)$  be a complete graph (i.e.,  $K_n$ ) with vertex set  $V = \{1, \dots, n\}$  and edge set  $E = \{e_x \mid x = (n(n-1) - ((i-n+1)(i-n) + (2n-j))) / 2$  where  $i < j$  are in  $V$  and  $e_x$  is incident with  $i$  and  $j\}$ . Thus,  $\text{Card}(V) = n$  and the  $\text{Card}(E) = n(n-1)/2$ . Each edge is assigned a time dependent cost  $C(e_x, t)$  which is equal to the cost of electing to construct  $e_x$  at time  $t$ . These costs are known and can be viewed as an  $n(n-1)/2$  by  $n-1$  cost matrix. A spanning tree for  $G$  consists of the vertex set  $V$  together with a subset of edge set  $E$  so that all of the vertices are connected and there are no cycles[10]. A ordered spanning tree for  $G$  also contains an ordering of the tree edges. We will assume that an edge is constructed per time interval. The total cost of a spanning tree with time dependent edge costs is simply the sum of the costs of each edge depending on the order (i.e., time) that the edge was constructed. In other words, the cost is evaluated by summing the costs of each edge in the ordered spanning tree in accordance with a given cost matrix that holds the cost of constructing any edge  $e_x$  (where  $1 \leq x \leq n(n-1)/2$ ) at time  $t$  (where  $1 \leq t \leq n-1$ ). The problem here is to find a minimal spanning tree(s) with a minimum cost[7, 19]. Since a complete graph on  $n$  vertices has  $n^{n-2}$  spanning trees and each of these has  $n-1$  edges, there are a total of  $(n^{n-2})(n-1)!$  edge ordered spanning trees.

### ***Genetic Algorithm Methodology***

A **genetic algorithm (GA)** is a biologically inspired, highly robust heuristic search procedure that can be used to find optimal (or near optimal) solutions to NP hard problems. The GA paradigm uses an adaptive methodology based on the ideas of Darwinian natural selection and genetic inheritance on a population of potential solutions. It employs the techniques of crossover (or mating), mutation, and survival of the fittest to generate new, typically fitter members of a population over a number of generations [1, 2, 3, 18].

We propose GAs for solving this optimal sequencing problem using novel multiple genomic redundant encoding schemes. Our GAs create and evolve an encoded population of potential solutions so as to facilitate the creation of new *feasible* members by standard mating and mutation operations. (A feasible search space contains only members which satisfy the problem constraints, that is, a sequencing [4, 5, 6, 7, 13, 14].) When feasibility is not guaranteed,

numerous methods for maintaining a feasible search space have been addressed in [11, 18], but most are elaborate and complex. They include the use of problem-dependent genetic operators and specialized data structures, repairing or penalizing infeasible solutions, and the use of heuristics.) By making use of problem-specific encodings, our problem insures a *feasible* search space during the classical operations of crossover and mutation and, in addition, eliminates the need to screen during the generation of the initial population.

We adapted many of the standard GA techniques found in [1, 2, 3] to this problem. A brief description of these techniques follows. Selection of parents for mating involves randomly choosing one very fit member of the population (i.e., one with a small total cost) and the other member randomly. The reproductive process is a simple crossover operation whereby two randomly selected parents are cut into sections at some randomly chosen positions and then have the parts of their encodings swapped to create two offspring (children). In our application the crossover operation produces an encoding for the offspring that have element values that always satisfy the position bounds (i.e., range constraints). Mutation is performed by randomly choosing a member of the population, cloning it, and then changing values in its encoding at randomly chosen positions subject to the range constraints for that position. A grim reaper mechanism replaces low scoring members in the population with newly created more fit offspring and mutants. Our fitness measure will be the total cost of the spanning tree.. The GA is terminated when, for example, either no improvement in the best fitness value is observed for a number of generations, a certain number of generations have been examined, and/or a satisficing solution is attained (i.e., the result is not necessarily optimum, but is satisfactory).

### **The Generic Genetic Algorithm**

We can now state the generic genetic algorithm we used for this application:

- 1) Randomly initialize a population of multiple genomic redundantly encoded potential solutions.
- 2) Map each population member to its equivalent phenome.
- 3) Calculate the fitness of any population member not yet evaluated.
- 4) Sort the members of the population in order of fitness.
- 5) Randomly select parents for mating and generate offspring using crossover.
- 6) Randomly select and clone members of the population to generate mutants.
- 7) Sort all the members of the expanded population in order of fitness adjusting each of the multiple segments to reflect the phenome with best fit.
- 8) Use the grim reaper to eliminate the population members with poor fitness.
- 9) If (termination criteria is met) then return best population member(s)  
else go to step 5.

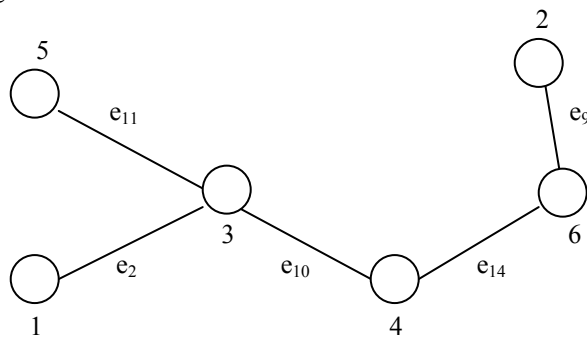
***Fitness***

The fitness of a population member is evaluated by summing the costs of each edge in the ordered spanning tree in accordance with the given cost matrix that holds the cost of constructing any edge  $e_x$  (where  $1 \leq x \leq n(n-1)/2$ ) at time  $t$  (where  $1 \leq t \leq n-1$ ).

Here is an example with a cost matrix for  $K_5$  and three different edge ordered spanning trees:

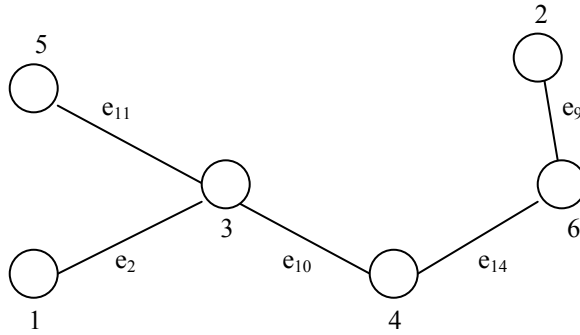
	t = 1	t = 2	t = 3	t = 4	t = 5
$e_1$	13	14	15	16	17
$e_2$	2	5	4	3	5
$e_3$	6	10	4	10	18
$e_4$	6	7	7	8	9
$e_5$	7	7	7	7	7
$e_6$	6	5	4	3	6
$e_7$	12	15	14	13	15
$e_8$	6	9	12	13	15
$e_9$	4	3	5	5	2
$e_{10}$	6	9	9	9	4
$e_{11}$	8	9	1	7	6
$e_{12}$	7	8	3	7	4
$e_{13}$	4	5	6	7	8
$e_{14}$	5	2	5	2	5
$e_{15}$	6	8	10	12	14

figure 1



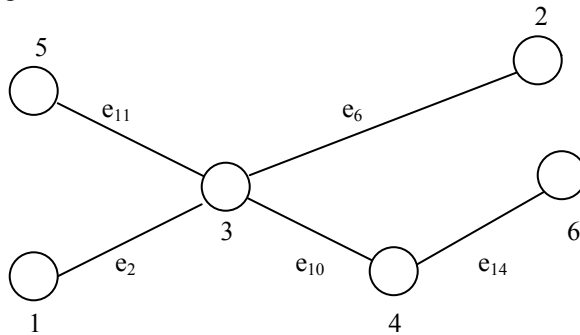
multiple genome for the edge ordered spanning tree  $e_{11} e_2 e_9 e_{14} e_{10}$   
 with cost = 24 3 6 3 4 3 6 4 3 2 9 10 11 14 2 4 3 1 1 3 1 2 2 1 4 1 5 3 2

figure 2



multiple genome for the edge ordered spanning tree  $e_9 e_{14} e_{11} e_{10} e_2$   
 with cost = 16 3 6 3 4 3 6 4 3 2 9 10 11 14 5 1 3 2 1 2 2 2 1 1 2 5 4 3 1  
 note that this is the same spanning tree with a different edge ordering.

figure 3



multiple genome for a minimal edge ordered spanning tree  $e_2 e_{14} e_{11} e_6 e_{10}$   
 with cost = 10 3 3 3 4 3 3 3 4 2 6 10 11 14 1 4 3 2 1 2 2 3 1 1 1 5 4 2 3

### Encodings

This application has multiple tree encodings to identify the spanning tree via different representations. Here we define the prufer code, neville code, and edge code for a spanning tree on  $K_n$ .

The spanning tree  $e_2 e_9 e_{10} e_{11} e_{14}$  can represent itself. This is one of multiple representations. The edge array  $E$  is of size  $n - 1$ . We call this the **edge code** and let  $E[1] = 2$ ,  $E[2] = 9$ ,  $E[3] = 10$ ,  $E[4] = 11$ , and  $E[5] = 14$  (or 2 9 10 11 14).

Any spanning tree on  $V = \{ 1, 2, \dots, n \}$  can also be encoded by an array of size  $n - 2$  where the value of any position can range over the values  $1, \dots, n$ .

An encoding of a spanning tree can also be represented as an array  $PR$  (**prufer coding**) where  $1 \leq PR[k] \leq n$  for  $1 \leq k \leq n$ . Prufer's method encodes a tree by iteratively deleting the leaf node with the smallest label and recording its unique neighbor in the next position of the  $PR$  array until only one edge remains.

Consider an example, with  $n = 6$  and  $PR[1] = 3$ ,  $PR[2] = 6$ ,  $PR[3] = 3$ , and  $PR[4] = 4$  (or 3634) which represents the spanning tree in figure 1 above.

Given a prufer array  $PR$  the tree can be constructed as follows. The reverse process begins with  $L$ , the set of all labels **in ascending order** that do not appear in the code. Since a code of length  $n - 2$  encodes a tree with  $n$  nodes, the integers  $1, \dots, n$  are used. Iteratively consider each label in the code starting from the first position and sequentially moving to the last. Let  $u$  be the current label in the code being considered and  $v$  be the smallest label in  $L$  then add the edge connecting  $u$  and  $v$  to the tree and remove  $v$  from  $L$ . If  $u$  is not in any other higher position in the code, add  $u$  to  $L$  so that the entries in  $L$  are **still in order**. Finally, after the last position of the code has been considered  $L$  should have exactly two labels left  $u$  and  $v$  so add the edge connecting  $u$  and  $v$  to the tree.

Another encoding of a spanning tree can also be represented as an array  $N$  (**neville coding**) where  $1 \leq N[k] \leq n$  for  $1 \leq k \leq n - 2$ . Neville's prufer-like method also begins by deleting the leaf node with the smallest label and recording its unique neighbor  $v$  in the next position of the  $N$ , if however,  $v$  is a leaf in the resulting subtree then  $v$  is considered next, else the leaf in the resulting tree with the smallest label is considered. Like prufer, this continues until only one edge remains.

Consider an example, with  $n = 6$  and  $N[1] = 3$ ,  $N[2] = 6$ ,  $N[3] = 4$ , and  $N[4] = 3$  (or 3643) which again represents the spanning tree in figure 1 above.

Given a neville array  $N$  the tree can be constructed as follows. The reverse process begins with  $L$ , the set of all labels in order that do not appear in the code. Since a code of length  $n - 2$  encodes a tree with  $n$  nodes, the integers  $1, \dots, n$  are used. Iteratively consider each label in the code starting from the first position and sequentially moving to the last. Let  $u$  be the current label in the

code and  $v$  be the smallest label in  $L$  then add the edge connecting  $u$  and  $v$  to the tree and remove  $v$  from  $L$ . If  $u$  is not in any other higher position in the code, add  $u$  to **the front** of  $L$ . Finally, after the last position of the code has been considered  $L$  should have exactly two labels left  $u$  and  $v$  so add the edge connecting  $u$  and  $v$  to the tree.

This application also has multiple permutation encodings to identify the sequencing (ordering) via different representations. Here we define the permutation code, forward code, and backward code for a permutation.

The 5-permutation 41532 or  $P[1] = 4, P[2] = 1, P[3] = 5, P[4] = 3,$  and  $P[5] = 2$  can represent itself. This is one of multiple representations of 41532. We call this the **permutation code** and  $PC[1] = 4, PC[2] = 1, PC[3] = 5, PC[4] = 3,$  and  $PC[5] = 2$ .

An  $n$  permutation of the integers  $\{ 1, 2, \dots, n \}$  can also be encoded by an array of size  $n$  where the value of the  $k^{\text{th}}$  position can range over the values  $1, 2, \dots, n-k+1$ .

An encoding of a permutation of the elements can also be represented as an array  $FC$  (**forward coding**) where  $1 \leq FC[k] \leq n-k+1$  for  $1 \leq k \leq n$ . In order to decode a permutation code  $FC$  to obtain the permutation that it represents, begin with an empty array  $P$  of size  $n$ , then for  $1 \leq i \leq n$  fill in the  $FC[i]^{\text{th}}$  empty position (from left to right starting at position 1) of  $P$  with the value  $i$ .

Consider an example, with  $n = 5$  and  $FC[1] = 2, FC[2] = 4, FC[3] = 3,$   $FC[4] = 1,$  and  $FC[5] = 1$  (or 24311) which represents the permutation  $P[1] = 4,$   $P[2] = 1, P[3] = 5, P[4] = 3,$  and  $P[5] = 2$  (or 41532).

Given a permutation array  $P$ , the reverse process begins with an empty array  $FC$  of size  $n$ , then for  $1 \leq i \leq n$  starting with  $i = 1$  and ending with  $i = n$  fill in the  $i^{\text{th}}$  position of  $FC$  (from left to right starting at position 1) with the value  $k - (\# \text{ of values } \leq i \text{ that occur before position } i \text{ in } P)$  where  $P[k]$  contains the value  $i$ . (Note that  $FC[n]$  will always be 1, so that, we can shorten  $FC$  to an  $n - 1$  element array if we wish.) An ordering of a set of 5 elements  $\{ e_1, e_2, e_3, e_4, e_5 \}$  based on the forward code (24311) would then be 5-tuple  $(e_4, e_1, e_5, e_3, e_2)$ .

An encoding of a permutation of the elements can also be represented as an array  $BC$  (**backward coding**) where  $1 \leq BC[k] \leq n-k+1$  for  $1 \leq k \leq n$ . In order to decode a permutation code  $BC$  to obtain the permutation that it represents, begin with an empty array  $P$  of size  $n$ , then for  $1 \leq i \leq n$  fill in the  $BC[i]^{\text{th}}$  empty position (from left to right starting at position 1) of  $P$  with the value  $n-i+1$ .

Consider an example, with  $n = 5$  and  $BC[1] = 3$ ,  $BC[2] = 1$ ,  $BC[3] = 2$ ,  $BC[4] = 2$ , and  $BC[5] = 1$  (or 31221) which represents the permutation  $P[1] = 4$ ,  $P[2] = 1$ ,  $P[3] = 5$ ,  $P[4] = 3$ , and  $P[5] = 2$  (or 41532).

Given a permutation array  $P$ , the reverse process begins with an empty array  $BC$  of size  $n$ , then for  $1 \leq i \leq n$  starting with  $i = 1$  and ending with  $i = n$  fill in the  $i^{\text{th}}$  position of  $BC$  (from left to right starting at position 1) with the value  $k - (\# \text{ of values } \geq i \text{ that occur before position } i \text{ in } P)$  where  $P[k]$  contains the value  $n - i + 1$ . (Note that  $BC[n]$  will always be 1, thus, we can shorten  $BC$  to an  $n - 1$  element array if we wish.) An ordering of a set of 5 elements  $\{ e_1, e_2, e_3, e_4, e_5 \}$  based on the backward code (31221) would then be 5-tuple  $(e_4, e_1, e_5, e_3, e_2)$ .

Next we consider a multiply redundant representation [12] that can be given by concatenating these lists. Thus a **multiply redundant representation** of the spanning tree in figure 2 would then be

3 6 3 4 3 6 4 3 2 9 10 11 14 5 1 3 2 1 2 2 2 1 1 2 5 4 3 1

with prufer, nevelle, edge, forward, backward, and permutation codes concatenated in that order. It is easy to mate and mutate this multiple representation scheme [6, 7, 13, 14, 19], however the resulting list may not reflect the same phenotype in each segment of the multiple genome. In this case we simply choose a best performing segment combination and repair the entire multiple genome to mirror the best phenome in all of the other redundant segments. Suppose

3 6 3 4 3 6 4 3 2 9 10 11 14 5 1 3 2 1 2 2 2 1 1 2 5 4 3 1 is parent 1 and  
4 3 1 2 4 3 1 2 1 2 3 9 10 3 2 2 2 1 1 4 3 1 1 5 2 1 3 4 is parent 2 then  
mating on positions

0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0  
swap positions to get the child

3 3 3 4 4 6 1 3 2 3 9 10 11 3 1 3 2 1 2 2 3 1 1 2 5 4 1 3  
after swapping in those positions.

(Notice in last segment we get 25413 since those positions reflect 1 and 3 in the first genome 25431 in the same order as the second genome 52134.)

Assuming the first segment 3334 and the fifth 22311 represent the best phenome combination, we repair the entire code and get

3 3 3 4 3 3 3 4 2 6 10 11 14 1 4 3 2 1 2 2 3 1 1 1 5 4 2 3

which is the multiple genome for figure 3.

## Results

We experimented with all possible combinations of the codes. The multiple genomic redundant representation **using all the segments was most**

**efficient.** The experiments using all codes were consistently most efficient even though more computational overhead was generated.

The multiple genomic redundant representation using all the segments was also examined as to which representations dominated at various stages of the search. The **forward and backward as well as the prufer-like codes were used extensively in the earlier generations** when the GA was searching more globally since these coding scheme do not have a good locality property. **In the later stages the GA adapted its search using mostly the permutation and edge codes** that have a stronger locality property.

### *Conclusions*

We considered using multiple genomic redundant representations in a self-adapting genetic algorithm to solve the minimal edge-ordered spanning tree problem that is NP hard. We then demonstrated that using multiple genomic redundant representations to create a self-adapting genetic algorithm by employing various codes with different locality properties improves the genetic algorithm's efficiency. The GA solving this NP hard problem, employed non-locality or locality representations when appropriate since the GA adapts to its current search needs making the GA more efficient.

### *Acknowledgements*

We wish to thank Pace University's Seidenberg School of Computer Science and Information Systems for partially supporting this research.

### *References*

- [1] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, (2001).
- [2] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, (1989).
- [3] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, (1991).
- [4] M. L. Gargano and S. C. Friederich, On Constructing a Spanning Tree with Optimal Sequencing, Congressus Numerantium 71, (1990) pp. 67-72.
- [5] M. L. Gargano, L. V. Quintas and S. C. Friederich, Matroid Bases with Optimal Sequencing, Congressus Numerantium 82, (1991) pp. 65-77.

- [6] M. L. Gargano and W. Edelson, A Genetic Algorithm Approach to Solving the Archaeology Seriation Problem, *Congressus Numerantium* 119, (1996) pp. 193-203.
- [7] W. Edelson and M. L. Gargano, Minimal Edge-Ordered Spanning Trees Solved By a Genetic Algorithm with Feasible Search Space, *Congressus Numerantium* 135, (1998) pp. 37-45.
- [8] F. S. Roberts, *Discrete Mathematical Models*, Prentice-Hall Inc., (1970).
- [9] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, Holden-Day Inc. (1968).
- [10] K. H. Rosen, *Discrete Mathematics and Its Applications*, Fourth Edition, Random House (1998).
- [11] Z. Michalewicz, Heuristics for Evolutionary Computational Techniques, *Journal of Heuristics*, vol. 1, no. 2, (1996) pp. 596-597.
- [12] F. Rothlauf and D.E. Goldberg, Redundant Representations in Evolutionary Computation, *Evolutionary Computation*, vol. 11, no. 4, 2003 pp.381-416.
- [13] J. DeCicco, M.L. Gargano, W. Edelson, A Minimal Bidding Application (with slack times) Solved by a Genetic Algorithm Where Element Costs Are Time Dependent, *GECCO*, (2002).
- [14] M.L. Gargano, W. Edelson, Optimally Sequenced Matroid Bases Solved By A Genetic Algorithm with Feasible Search Space Including a Variety of Applications, *Congressus Numerantium* 150, (2001) pp. 5-14.
- [15] M.L. Gargano, Maheswara Prasad Kasinadhuni , Self-adaption in Genetic Algorithms using Multiple Genomic Redundant Representations, *Congressus Numerantium* 167, (2004) pp. 183-192.
- [16] E.H. Neville, The Codifying of Tree Structure, *Proceedings of the Cambridge Philosophical Society*, vol. 149, (1953) pp.381-385.
- [17] N. Deo, P. Micekivius, Prufer-like Codes for Labeled Trees, *Congressus Numerantium* 151, (2001) pp. 65-73.
- [18] Z. Michalewicz, D.B. Fogel, *How to Solve It: Modern Heuristics*, 2<sup>nd</sup> ed. Springer-Verlag, (2004).
- [19] M. L. Gargano and W. Edelson, Constrained Minimal Edge-Ordered Spanning Trees Solved By a Genetic Algorithm with Feasible Encodings, *Congressus Numerantium* 143, (2000) pp. 5-21.