Encoding Bounded-Diameter Spanning Trees with Permutations and with Random Keys

Bryant A. Julstrom

Department of Computer Science
St. Cloud State University, St. Cloud, MN, 56301 USA
julstrom@eeyore.stcloudstate.edu

Abstract. Permutations of vertices can represent constrained spanning trees for evolutionary search via a decoder based on Prim's algorithm, and random keys can represent permutations. Though we might expect that random keys, with an additional level of indirection, would provide inferior performance compared with permutations, a genetic algorithm that encodes spanning trees with random keys is as effective as one whose genotypes are permutations of vertices in comparisons on a variety of instances of the bounded-diameter minimum spanning tree problem. These results suggest that either coding may be used, at the programmer's convenience, in evolutionary algorithms for problems involving constrained spanning trees.

1 Introduction

Evolutionary algorithms often search spaces of permutations. However, when simple positional evolutionary operators like k-point crossover and position-by-position mutation are applied to permutations, the offspring they build are generally not permutations, so researchers must apply specialized operators. A large number of these, particularly crossovers, have been developed [1].

Bean [2] proposed an indirect representation of permutations for evolutionary search to which positional operators can be applied. He called this representation random keys. In it, a genotype is a sequence of floating-point values, called keys and usually falling between 0.0 and 1.0, associated with the items to be ordered. Sorting the keys yields a permutation of the items; this is the permutation the genotype represents. For example, if the integers 0 through 9 label ten items, then the key sequence

$$(0.48, 0.66, 0.07, 0.33, 0.38, 0.72, 0.88, 0.54, 0.25, 0.42)$$

represents this permutation of those items: (2 8 3 4 9 0 7 1 5 6). Every sequence of keys represents a valid permutation, so simple operators like k-point crossover and position-by-position mutation always yield valid genotypes.

Rothlauf [3, pp.180-182] has argued that random keys have good properties for evolutionary search. Researchers have used them to represent permutations, in turn representing candidate solutions, in evolutionary algorithms for a variety

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1272–1281, 2004.

of problems, including machine scheduling, vehicle routing, and the quadratic assignment problem [2], constrained facility layout problems [4], deceptive ordering problems [5] [6], cellular manufacturing [7], and job shop scheduling [8].

Rothlauf, Goldberg, and Heinzl [9] [3, pp.178–190] described a random-key representation of spanning trees called Network Random Keys, or NetKeys. Given a connected, undirected graph G, NetKeys associates keys with G's edges. Sorting the keys yields a permutation of the edges, and a decoding algorithm based on Kruskal's algorithm [10] examines the edges in their permuted order to build a spanning tree on G. The decoder can enforce constraints on the spanning tree, such as a bound on its vertices' degrees. If G has n vertices, it may have up to $\binom{n}{2}$ edges, so the size of each NetKeys genotype is $O(n^2)$, and the time required to sort it—and impose an ordering on G's edges—is $O(n^2 \log n)$.

Permutations of a graph's vertices can also represent spanning trees, via a decoder based on Prim's algorithm [11], and these permutations can in turn be represented by random keys. Each of these codings offers advantages and disadvantages of implementation; does either provide better performance?

Permutations of vertices and random keys, with operators appropriate to them, were implemented in generational genetic algorithms and compared on 25 instances of the bounded-diameter minimum spanning tree problem, which Section 2 describes. These trials revealed no significant or consistent differences in the performance of the two codings. This suggests that either coding may be used, at the programmer's convenience, in evolutionary algorithms for problems involving constrained spanning trees.

Following the description of the bounded-diameter minimum spanning tree problem, this paper presents permutation and random-key codings of bounded-diameter spanning trees, genetic algorithms that use them, and the comparisons of the two genetic algorithms.

2 The Bounded-Diameter MST Problem

In a tree, the eccentricity of a vertex v is the maximum number of edges on any path from v to another vertex. The diameter of a tree is the maximum eccentricity of its vertices, thus the maximum number of edges along any path in it. The center of a tree is the single vertex (if its diameter is even) or the two vertices (if odd) of minimum eccentricity, and a vertex's depth is the number of edges on the path from it to the tree's center.

Given a connected, undirected graph G on n vertices and a bound D, a bounded-diameter spanning tree (BDST) is a spanning tree on G with diameter no greater than D. Figure 1 shows two BDSTs on $n = 20$ vertices. One has even diameter; the vertex v is its center. The other has odd diameter; the vertices v and w together form its center.

If weights are associated with G's edges, then a bounded-diameter minimum spanning tree (BDMST) is a BDST on G of minimum total weight. The search for such a tree is the bounded-diameter minimum spanning tree problem. This problem is NP-hard for $4 \leq D < n - 1$ [12, p.206], though it is solvable in

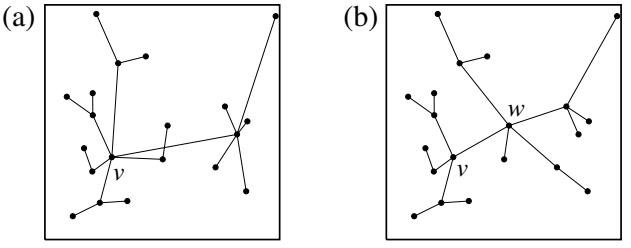

Fig. 1. (a) A tree on 20 vertices with diameter 4; v is its center. (b) A tree on the same vertices with diameter 5; v and w together form its center

polynomial time if $D \leq 3$ or if all the edge weights in G are equal. The bounded-diameter minimum spanning tree problem has applications in a variety of areas, including communications network design [13], mutual exclusion in distributed systems [14], and data compression [15],

Abdalla, Deo, and Gupta [16] [17] described two heuristics for this problem, one of which imitates Prim's algorithm [11] but eschews edges whose inclusion would violate the diameter bound. It maintains the lengths of all paths and the eccentricities of all vertices in the growing BDST, and it requires time that is $O(n^3)$.

More recently, Raidl and Julstrom [18] described a Prim-based, greedy heuristic that builds a BDST beginning at its center. It avoids edges that would place vertices at depths greater than $\lfloor D/2 \rfloor$—that is, more than $\lfloor D/2 \rfloor$ edges from the center—so that no two vertices are more than D edges apart. This algorithm requires time that is $O(n^2)$, and it underlies the decoding algorithms of the two codings of BDSTs that the next section describes.

3 Encoding Bounded-Diameter Spanning Trees

Spanning trees can be encoded for evolutionary search in a variety of ways [19] [3, pp.128–197], most of which can be adapted to represent bounded-diameter spanning trees. Our focus, however, is particularly on permutations and random keys as codings of BDSTs. This section describes these codings and operators appropriate to them.

3.1 With Permutations

Julstrom and Raidl [20] encoded bounded-diameter spanning trees on a graph G as permutations of G's vertices. Evaluating a permutation requires making explicit the tree it represents; the following decoder is based on Prim's algorithm.

Let c[·] be a permutation of G's vertices. The first vertex (if the diameter bound D is even) or the first two vertices (if D is odd) that c[·] lists form the center of c[·]'s tree. The decoder appends the remaining vertices to the tree in their order in c[·]. The depth of each new vertex is one greater than the depth

of the vertex in the tree to which it is joined, so each new vertex is attached by its lowest-weight edge to a vertex of degree less than $\lfloor D/2 \rfloor$. Then no vertex has depth greater than $\lfloor D/2 \rfloor$, and the tree's diameter does not exceed D.

Figure 2 summarizes the permutation decoding algorithm. A vertex's depth is fixed when it joins the tree and does not change thereafter, but as in Prim's algorithm, the nearest-connection information for the remaining unconnected vertices must be updated after each vertex joins the tree, so the decoder's time is $O(n^2)$. The space a permutation requires is $O(n)$.

```
T ← ∅
v_o ← c[0]
U ← V − {v_o}
C ← {v_o}
depth[v_o] ← 0
if D is odd
    v_1 ← c[1]
    T ← {(v_o, v_1)}
    U ← U − {v_1}
    C ← C ∪ {v_1}
    depth[v_1] ← 0
while U ≠ ∅ do
    u ← the next vertex in c[·]
    v ← the vertex in C nearest u
    T ← T ∪ {(u, v)}
    U ← U − {u}
    depth[u] ← depth[v] +1
    if depth[u] < ⌊D/2⌋
        C ← C ∪ {u}
return T
```

Fig. 2. The decoding algorithm for the permutation coding of bounded-diameter spanning trees. c[·] is the permutation, T is the tree's edge set, V is the graph's vertex set, U is the set of unconnected vertices, and C is the set of connected vertices to which a new edge may be connected without violating the diameter bound

A random permutation can be generated in time that is $O(n)$. An appropriate crossover operator is Reeves' C1 [21] (also described, with different names, by Smith [22] and Prosser [23]). C1 chooses a crossover point at random and copies one parent into the offspring up to that point. It then copies the remaining values in order from the second parent into the offspring's remaining positions. This operator usually preserves the center of the first parent. Its time is $O(n)$.

A mutation operator swaps the vertices at two random positions in the parent genotype, thus exchanging the times at which the decoder includes those vertices

in the tree. Identifying the two positions and exchanging their contents require only constant time, but copying the parent genotype into the offspring is $O(n)$.

3.2 With Random Keys

To use random keys to represent bounded-diameter spanning trees, concatenate the discussions of random keys and permutations of vertices. In particular, let G be a graph with n vertices and let D be the diameter bound. A genotype is a sequence of n random keys in $[0.0, 1.0]$, one for each vertex in G. To identify the BDST a genotype represents, sort the keys to obtain a permutation of G's vertices. Then, as in Section 3.1, the Prim-based decoder builds the tree with diameter no more than D corresponding to the permutation.

The time required to sort a random-key genotype is $O(n \log n)$ and the time to identify the tree from the resulting permutation is $O(n^2)$. Thus the time of the entire decoding process is $O(n^2)$. The space a genotype requires is again $O(n)$.

With random keys, appropriate operators are two-point crossover and position-by-position mutation: with a small probability, each value from the parent genotype is replaced by a new random value in $[0.0, 1.0]$. These operators' times are linear in n.

3.3 Two Genetic Algorithms

The permutation and random-key codings of bounded-diameter spanning trees were implemented in two genetic algorithms for the BDMST problem. The GAs are generational and initialize their populations with random genotypes, and they select parents in tournaments. Both generate new genotypes from those parents using either crossover or mutation, never both; each offspring is generated by exactly one operator. Both GAs run through fixed numbers of generations.

4 Comparisons

The permutation-coded GA and the random-key-coded GA were compared on 25 Euclidean instances of the bounded-diameter minimum spanning tree problem, five instances each of $n = 50$, 70, 100, 250, and 500 points. The instances are found in Beasley's OR-Library[1] [24], where they are listed as instances of the Euclidean Steiner problem. The tests used the first five instances of each size; the library contains fifteen instances of each size (and others).

Each instance consists of random points in the unit square. The points are treated as the vertices of complete graphs whose edge weights are the Euclidean distances between the points. When $n = 50$, the diameter bound D was set to 5; when $n = 70$, $D = 7$; when $n = 100$, $D = 10$; when $n = 250$, $D = 15$, and when $n = 500$, $D = 20$.

[1] `mscmga.ms.ic.ac.uk/info.html`

The two EAs' common parameters were set identically. For a BDMST problem instance with n points, their population sizes were $20\sqrt{n}$ and their numbers of generations in each run were $50\sqrt{n}$, so that a run always generated $1000n$ new genotypes. Both algorithms selected parent genotypes in tournaments of size two, without replacement, and a tournament's winner always became a parent. Both applied crossover with probability 60%, and mutation, therefore, with probability 40%. The random-key-coded GA mutated each position in its genotypes with probability $3/n$. Table 1 lists the population sizes and run lengths for the two GAs and the mutation probabilities in the random-key-coded GA.

Table 1. Population sizes and numbers of generations per run of the two genetic algorithms, and the probability of mutating one symbol in the random-key-coded GA. On a problem instance of n vertices, the total number of new genotypes is always $1000n$, with the algorithms' population sizes set to $20\sqrt{n}$ and their numbers of generations to $50\sqrt{n}$. In the random-key-coded GA, P[mu] $= 3/n$

n	PopSize	Generations	P[mu]
50	141	353	0.060
70	167	418	0.043
100	200	500	0.030
250	316	790	0.012
500	447	1118	0.006

On each instance, both EAs were run 50 independent times. Table 2 summarizes the results of these trials. For each instance, the table lists the number of points n and the diameter bound D. For each GA and each instance, it lists the length of the shortest bounded-diameter tree found and the mean and standard deviation of the 50 trials' tree lengths. The smaller best values and the smaller mean values for each instance are bold.

Because of the additional level of indirection imposed by random keys, we might expect the GA using them to be less effective than the permutation-coded GA, but this is not the case. There are no significant or consistent differences between the performance of the permutation-coded GA and that of the random-key-coded GA on these BDMST problem instances. For example, the two GAs return equal best tree lengths on four of the five instances with $n = 50$ vertices, and on the remaining instance the values differ by less than 0.15%. The permutation-coded GA returns the shorter mean tree lengths on three instances, and the random-key-coded GA on two.

Similarly, on the instances with $n = 100$ and $D = 10$, the permutation-coded GA identifies the shortest tree twice and returns the shorter mean tree length twice; the random-key-coded GA wins the remaining three contests of each kind. The results on the instances of other sizes are similar. No consistent winner emerges, and the differences between the best and mean tree lengths are always small compared to the respective standard deviations.

Table 2. For each set of 50 trials of the permutation-coded GA and the random-key-coded GA on each of the 25 BDMST problem instances: the length of the shortest tree found and the mean and standard deviation of the 50 tree lengths. The smaller best and mean values are bold.

Instance			Permutation-Coded GA			Random-Key-Coded GA		
n	D	num.	best	mean	stddev	best	mean	stddev
50	5	1	**7.602**	**7.838**	0.19	7.613	7.853	0.13
		2	**7.750**	7.895	0.12	**7.750**	**7.866**	0.10
		3	**7.251**	**7.477**	0.15	**7.251**	7.487	0.14
		4	**6.616**	**6.680**	0.09	**6.616**	6.706	0.10
		5	**7.388**	7.498	0.12	**7.388**	**7.489**	0.07
70	7	1	7.236	7.361	0.07	**7.234**	**7.358**	0.06
		2	**7.122**	**7.229**	0.08	**7.122**	7.247	0.07
		3	**6.987**	7.197	0.14	7.009	**7.151**	0.10
		4	**7.521**	7.646	0.10	**7.521**	**7.643**	0.07
		5	**7.269**	7.359	0.08	7.270	**7.343**	0.05
100	10	1	**7.818**	**7.919**	0.07	7.831	**7.919**	0.05
		2	7.873	**8.017**	0.08	**7.853**	8.043	0.09
		3	7.990	8.139	0.08	**7.982**	**8.137**	0.09
		4	8.009	8.143	0.07	**7.996**	**8.122**	0.06
		5	**8.193**	8.335	0.08	8.198	**8.313**	0.08
250	15	1	**12.440**	12.602	0.08	12.448	**12.580**	0.08
		2	12.237	12.432	0.10	**12.222**	**12.393**	0.10
		3	**12.117**	**12.282**	0.08	12.178	12.315	0.07
		4	**12.572**	12.824	0.11	12.632	**12.802**	0.07
		5	**12.358**	**12.608**	0.12	12.382	12.623	0.10
500	20	1	17.216	17.476	0.10	**17.156**	**17.429**	0.10
		2	**17.085**	17.311	0.11	17.097	**17.291**	0.10
		3	17.173	17.449	0.11	**17.164**	**17.369**	0.11
		4	**17.215**	17.484	0.13	17.266	**17.432**	0.09
		5	16.939	17.137	0.11	**16.872**	**17.092**	0.11

From these results we conclude that, in these genetic algorithms, the performances offered by the permutation coding and by the random-key coding are indistinguishable. More generally, these results suggest that either coding may be used to good effect on problems, like the BDMST problem, where a Prim-based decoding algorithm identifies the encoded spanning trees.

5 Conclusion

Given a connected, weighted, undirected graph G and a bound D, the bounded-diameter minimum spanning tree problem seeks a spanning tree on G of minimum weight among the trees with diameter no greater than D. Two indirect evolutionary codings of valid spanning trees represent BDSTs in genetic algorithms for this problem. Permutations of G's vertices encode trees via a decoding algorithm based on Prim's algorithm. Random keys are floating-point

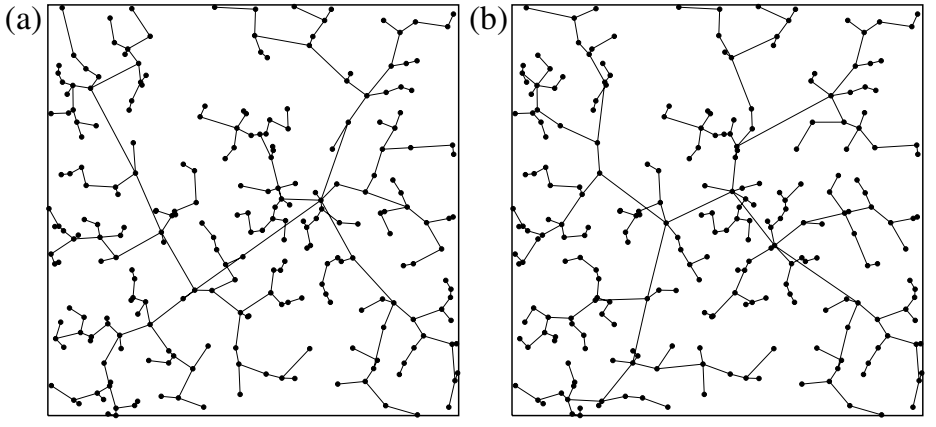

Fig. 3. On the first BDMST problem instance with $n = 250$ vertices and diameter bound $D = 15$, (a) the shortest tree found by the permutation-coded genetic algorithm; it has length 12.440; and (b) the shortest tree found by the random-key-coded GA; it has length 12.448

values associated with G's vertices; sorting a genotype of random keys yields a permutation of the vertices, which in turn represents a BDST.

Each coding has advantages and disadvantages. Permutations are more direct but require specialized variation operators. Random keys can be manipulated with traditional—and simple—operators, but decoding them requires the additional step of sorting. Both, however, always represent only valid trees; their decoding algorithms enforce the diameter bound. Also, both are more efficient in space and time than Network Random Keys, which associate floating-point values with the underlying graph's edges.

Implementing the two codings in generational genetic algorithms for the BDMST problem and comparing them on a range of Euclidean instances revealed no significant or consistent differences in the performance they offer. Either coding may be used at their authors' convenience in evolutionary algorithms for constrained spanning tree problems that, like the BDMST problem, admit a coding via permutations.

It is undoubtedly possible to improve the absolute performance of the genetic algorithms described here. The permutation-coded GA might benefit from the replacement of the C1 crossover operator with another of the many crossovers that have been described for permutations. The random-key-coded GA's genotypes might be augmented with ES-style mutation parameters, as Schindler [25] has suggested. Both might be more effective with different sets of parameter values like population size and operator probabilities.

References

1. Whitley, D.: Permutations. In Bäck, T., Fogel, D.B., Michalewicz, Z., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Philadelphia (2000) 274–284 In Chapter 33, Recombination.
2. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing **6** (1994) 154–160
3. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Volume 104 of Studies in Fuzziness and Soft Computing. Physica-Verlag, Heidelberg (2002)
4. Norman, B.A., Smith, A.E.: Random keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems. In: Proceedings of 1997 IEEE International Conference on Evolutionary Computation, IEEE, IEEE Neural Network Council, Evolutionary Programming Society, IEEE (1997) 407–411
5. Knjazew, D., Goldberg, D.E.: OMEGA - Ordering messy GA: Solving permutation problems with the fast messy genetic algorithm and random keys. Technical Report 2000004, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2000)
6. Bosman, P.A.N., Thierens, D.: Permutation optimization by iterated estimation of random keys marginal product factorizations. In Guervós, J.J.M. et al., eds: Parallel Problem Solving from Nature – PPSN VII. Volume 2439 of LNCS., Berlin, Springer-Verlag (2002) 331–340
7. Gonçalves, J.F., Resende, M.G.C.: A hybrid genetic algorithm for manufacturing cell formation. Technical report, AT&T Labs (2002) TD-5FE6RN.
8. Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. Technical report, AT&T Labs (2002) TD-5EAL6J.
9. Rothlauf, F., Goldberg, D., Heinzl, A.: Network random keys – a tree network representation scheme for genetic and evolutionary algorithms. Technical Report 8/2000, University of Bayreuth (2000)
10. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematics Society **7** (1956) 48–50
11. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal **36** (1957) 1389–1401
12. Garey, M.R., Johnson, D.S.: Computers and Intractibility: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
13. Bala, K., Petropoulos, K., Stern, T.E.: Multicasting in a linear lightwave network. In: IEEE INFOCOM'93. (1993) 1350–1358
14. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems **7** (1989) 61–77
15. Bookstein, A., Klein, S.T.: Compression of correlated bit-vectors. Information Systems **16** (1991) 110–118
16. Abdalla, A., Deo, N., Gupta, P.: Random-tree diameter and the diameter constrained MST. Congressus Numerantium **144** (2000) 161–182
17. Deo, N., Abdalla, A.: Computing a diameter-constrained minimum spanning tree in parallel. In Bongiovanni, G., Gambosi, G., Petreschi, R., eds.: Algorithms and Complexity. Number 1767 in LNCS. Springer, Berlin (2000) 17–31
18. Raidl, G.R., Julstrom, B.A.: Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In Lamont, G. et al., eds: Proceedings of the 2003 ACM Symposium on Applied Computing, ACM Press (2003) 747–752

19. Raidl, G.R., Julstrom, B.A.: Edge-sets: An effective evolutionary coding of spanning trees. IEEE Transactions on Evolutionary Computation **7** (2003) 225–239
20. Julstrom, B.A., Raidl, G.R.: A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In Barry, A., ed.: 2003 Genetic and Evolutionary Computation Conference Workshop Program, Chicago, IL (2003) 2–7
21. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Computers and Operations Research **22** (1995) 5–13
22. Smith, D.: Bin packing with adaptive search. In Greffenstette, J.J., ed.: Proceedings of the First International Conference on Genetic Algorithms, Hillsdale, NJ, Lawrence Erlbaum (1985) 202–207
23. Prosser, P.: A hybrid genetic algorithm for pallet loading. In: Proceedings of the 8th European Conference on Artificial Intelligence, London, Pitman (1988)
24. Beasley, J.E.: OR-library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41** (1990) 1069–1072
25. Schindler, B., Rothlauf, F., Pesch, H.J.: Evolution strategies, network random keys, and the one-max-tree problem. In Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G.R., eds.: Applications of Evolutionary Computing. Volume 2279 of LNCS., Berlin, Springer-Verlag (2002) 143–152