New hybrid evolutionary algorithm for solving the bounded diameter minimum spanning tree problem

Sakshi Arora* and Garg M.L.
*School of Computer Science and Engineering, Shri Mata Vaishno Devi University, Katra (J&K), India,
sakshi@smvdu.ac.in, garg.ml@smvdu.ac.in

Abstract -Given a connected, weighted, undirected graph G and a bound D, the bounded diameter minimum spanning tree (BDMST) problem seeks a spanning tree on G of minimum weight among the trees in which no path between two vertices contains more than D edges. This problem is NP-hard for $4 \le D \le |v| - 1$. In present paper a new randomized greedy heuristic algorithm for solving BDMST is proposed. An evolutionary algorithm encodes spanning trees as lists of their edges, augmented with their center vertices. It applies operators that maintain the diameter bound and always generate valid offspring trees. These operators are efficient, so the algorithm scales well to larger problem instances. On 25 Euclidean instances of up to 1000 vertices, the EA improved substantially on solutions found by the randomized greedy heuristic.

Keywords- Evolutionary Algorithms, Bounded diameter spanning trees, greedy heuristic

Introduction

The bounded diameter minimum spanning tree (BDMST) problem is a combinatorial optimization problem that appears in many applications such as wire-based communication network design when certain aspects of quality of service have to be considered, in ad-hoc wireless network[1] and in the areas of data compression and distributed mutual exclusion algorithms [2]. The goal is to identify a tree-structured network of minimum costs in which the number of links between any pair of nodes is restricted by a constant D, the diameter. More formally, we are given an undirected connected graph G =(V;E) with node set V and edge set E and associated costs $c_e \ge 0$, for all e in E. We seek a spanning tree $T=(V,E_T)$ with edge set E_T being a subset of E whose diameter does not exceed $D \ge 2$, and whose total costs $c(T) = \sum_{e \in ET} c_e$ are minimal. This task can also be seen as choosing a center (one single node if D is even or an edge in the odd-diameter case)and building a height-restricted tree where the unique path from this center to any node of the tree consists of no more than H = D/2 edges. The BDMST problem is known to be NP-hard for $4 \le D \le V-1$. Techniques for solving the *BDMST* problem may be classified into two categories: exact methods and inexact (heuristic) methods. Exact approaches for solving the *BDMST* problem are based on mixed linear integer programming [3][4][5]. More recently, Gruber and Raidl suggested a branch and cut algorithm based on compact 0-1 integer linear programming [6]. However, being deterministic and exhaustive in nature, these approaches could only be used to solve small problem instances (e.g. complete graphs with less than 100 nodes). [7] presented a greedy heuristic algorithm - the One Time Tree Construction (*OTTC*) for solving the *BDMST* problem. *OTTC* is based on Prim's algorithm in [8]. It starts with a set of vertices, initially containing a randomly chosen vertex. The set is then repeatedly extended by adding a new vertex that is nearest (in cost) to the set, as long as the inclusion of the new node does not violate the constraint on the diameter of the tree. This algorithm is time consuming, and its performance is strongly dependent on the starting vertex. Raild and Julstrom [9] modified this approach to start from a predetermined centre and presented randomized greedy heuristic algorithm (RGH). RGH extends spanning tree from center by adding a randomize node from remain nodes and connecting it to node in the tree with smallest weight. *RGH* starts from a centre by randomly selecting a vertex and keeping it as the fixed center during the search. It then repeatedly extends the spanning tree from the center by adding a randomly chosen vertex from the remaining vertices, and connecting it to a vertex that is already in the tree via an edge with the smallest weight. The obtained results showed that on Euclidean instances *RGH* performs better than *OTTC*, whereas on non-Euclidean instances the situation is reversed. Raidl and Julstrom proposed a genetic algorithm for solving *BDMS*T problems which used edge-set coded [10] (*JR-ESEA*) and permutation-coded representations for individuals [11] (*JR-PEA*). Permutation coded evolutionary algorithms were reported to give better results than edge-set coded, but usually are much more time consuming. Another genetic algorithm, based on a random key representation, was derived in[12], sharing many similarities with the permutation-coded evolutionary algorithms. In[13], Gruber used four neighborhood types to implement variable neighborhood local search for solving the *BDMST* problem. They are: arc exchange neighborhood, level change neighborhood, node swap neighbourhood, and center change level neighborhood. Later, [14], re-used variable neighborhood searches as[13], embedding them in Ant Colony Optimization (*ACO*) and genetic algorithms for solving the *BDMST* problem. Both of their proposed algorithms (*ACO* and *GA*) exploited the neighborhood structure to conduct local search, to improve candidate solutions. In [15], Nghia and Binh proposed a new recombination operator which uses multiple parents to do the recombination in their genetic algorithm. Their proposed crossover operator helped to improve the minimum and mean weights of the evolved spanning trees. More recently, in [16], Alok and Gupta derived two improvements for *RGH* heuristics (given in [10]) and some new genetic algorithms for solving *BDMST* problems (notably the *GA* known as *PEA-I*). *PEA-I* employs a permutation-coded representation for individuals. In [17], Binh et al., also implement another variant of *RGH*, which is called *RGH*1. *RGH*1 is similar to *RGH*, except that when a new vertex is added to the expanding spanning tree, it is chosen at random, and connected to a randomly chosen vertex that is already in the spanning tree. Section 2 presents a discriminatory randomized greedy heuristic which is an improvement of RGH. Section 3 deals with the novel crossover operator proposed. Section 4 explains the novel GA strategy developed incorporating D-RGH as the initialization operator. Results are given in section 5. And finally, conclusion forms section 6.

The Discriminatory Randomized Greedy Heuristic (D-RGH)

As the starting vertex, significantly affects the weight of the generated trees therefore we propose not to choose the vertex randomly from the V. Our algorithm rather than choosing randomized center V (one center if D is even and two otherwise), we propose to choose the center from a subset C of vertices. In addition, while extending the tree also, the next vertex is not chosen randomly from V, but from a subset C of V. Note that the, spanning tree with diameter less than D has at least (n- 1)/(2*D) or (n-2)1(2*D) pendant vertices (i.e. vertices of degree 1) corresponding to the case of D being even or odd. Therefore, let P be the set of [2(n - 2)/D] vertices that are farthest from the center, these vertices will be connected in the end hoping that large edges will not have to be incorporated into the tree. They will be considered when remaining vertices have been connected.

Algorithm using path length matrix to determine the center:

CENTER_DET {}

Step C-1: {Determine path length matrix}

- Determine the Matrix X, each entry x[ij] of which is a number referring to the minimum number of edges between all pairs of i and j.

Step C-2: {Computing row total of X}

- Compute ∑ x[i] for all i. And let s[i] = ∑ x[i], the sum of lengths of shortest paths from vertex i to the rest (v-i) vertices.

Step C-3: {Sorting s[i]}

- Sort s[i] in the ascending order.

Step C-4: {Determine Q}

- Determine a number Q, chosen randomly between 2 and (i-1).

Step C-5: {Determining the center}

- Let C be the set of the first Q vertices from the sorted list s[i]. The center will be chosen from C.

Next, the algorithm D_RGH to generate the BDMST using the novel approach suggested by the authors is presented:

D_RGH {}

Step D-1: {Call subroutine CENTER_DET}

- Determine X[i,j] and s[i] for all i , j in V.
- Determine the set C.

Step D-2: {Fixing the center}

- Select $\{v_o\}$: a random vertex in C;
- C:= C- $\{v_o\}$; /* Update the set C */
- U := V- $\{v_o\}$; /* Update the set U of unconnected vertices */
- ST :=$\{v_o\}$; /* Add v_o to ST, the set of connected vertices added in the spanning tree */
- depth[v_o] := 0; /* Determine the distance of v_o from the center. */

Step D-3: {The odd diameter case}

- If D is odd then
- Select $\{v_1\}$: another random vertex in C;
- T := $\{(v_o, v_1)\}$; /* Add the edge (v_o, v_1) to the set of tree edges T*/
- U := U- $\{v_1\}$;
- ST := ST U $\{v_1\}$;
- C := C- $\{v_1\}$;
- depth[v_1] := 0;

Step D-4: { }

- Determine the set P /* P is the set of pendant vertices */
- U := U- P;

Step D-5: {Iterate for all vertices in U}

- While U ≠ Ø do
- v := randomize vertex in U;
- u := vertex in ST with min W(u,v);
- T := T U {(u, v)}; /* Update the tree edges*/
- U := U - {v}; /* Update the set U*/
- depth[v] := depth[u] + 1; /*Checking the diameter bound */
- If depth[v] < [D/2] then /* checking diameter bound*/

    ST: = ST U {v}; /* update the tree vertices*/

Step D-6: {Iterate for all vertices in P}

```
•  While P ≠ Ø do   /* P is set of farthest vertices*/
•  v := randomize vertex in R;
•  u := vertex in C which has minimum W(u,v);
•  T := T U {(u, v)};
•  P := P -{v};
•  depth[v] := depth[u] + 1;
•  If depth[v] < [D/2] then
           ST: = ST U {v};
```

Step D-7: {Return T}

Improved Genetic Algorithm

Genetic algorithm has proven effective on NP-hard problem. Much works research on NP-hard problem, particularly in problems relating to tree have been done. Several studies proposed representations for tree [16, 7, 9, 10, 12, 17]. This section proposes a new recombination operator (called multi-crossover operator) in genetic algorithm for solving BDMST problem.

A. Initialization

Use NRGH algorithm described above for initializing population and edge list for chromosome code.

B. Recombination operator

In genetic algorithm, recombination operator is used to produce a new child and it should be provided strong heritability. In traditional recombination operator, the child is produced from two parents but in proposed multi-crossover operator one, the child is produced from several parents. In BDMST problem, the child tree produced by recombination from several parent trees so it contains most of potential parental edges. While the traditional method works with two parents, our method will work with possible more than two parents in hope that the child could inherit much more potential edges. The operator may be considered superior as it preserves those edges in the parents which are encountered in more than two parents. And as the parents from which these edges are taken are themselves minimum spanning trees obtained from D-RGH, their edges should be passed on to the child (as in Elitism).

C. Mutation operator

We have used the edge delete mutation in our algorithm. The preliminary experiments show that the other mutation operators such as greedy edge replace mutation, center move mutation and subtree optimize mutation do not impact the tree significantly differently than the edge delete mutation.

GA_D-RGH{ }

```
Step 1: Set gen: =0;
Step 2: Call subroutine D-RGH for creating the bounded diameter spanning trees constituting the chromosomes of the population
Step3: P (t): = {T[1], T[2]….T[N]}, such that T[K] = {the edge set (u,v) for all u , v in V}
Step 4: /*Sketch of k-recombination operator*/
        T := Ø
Step 5:  /*Determine center*/
        If D is odd then
                $v_o$ := random center in T[i];
                U: = V- {$v_o$};
                C: = {$v_o$};
                depth[$v_o$] – 0
        else
                ($v_o$, $v_l$) - random center in T[i];
                T - {(vo, vi)};
                U- V- {vo, V,I;
                C – {vo, V- };
                depth[vo] - 0;
                depth[vi] - 0;
Step 6: while gen < $gen_{max}$ do
Step 7: Select{$T_1$,$T_2$,$T_3$…$T_k$}: = Φ (t); /* 2<k<N */
/* Φ = tournament operator */
Step 8: Crossover C: = $\Omega_c$ ($T_1$,$T_2$, $T_3$….$T_k$)
/* $\Omega_c$ = multi-crossover operator*/
Step 9: Mutate C ← $\Omega_m$(C)
/* $\Omega_m$ = mutation operator */
Step 10: Evaluate f(C)
Step 11: Evaluate P(t) := {f(T[1]),…….f(T[N])};
where f(T[i]) = ∑ fitness (u,v) where (u, v) is an edge between any two vertices in V
Step 12: If C ≡ any T ∈P(t) then discard C and go to step 7
else
Discard C.
Step 13: If f(C) < f(T[i]) then discard C and go to step 8
Step 14: end if
Step 15: end if
Step 14: gen ← gen+1
Step 19: end while
Step 20: return C, f(C)
```

Experiment and Evaluation

The EA Framework

The representation and operators that the last section described were implemented in a conventional steady-state evolutionary algorithm. The EA applies the randomized greedy heuristic to generate candidate solutions for its initial population, so it starts with a diverse collection of relatively good solutions. It selects parents in tournaments with replacement. Recombination generates some offspring, but every offspring is mutated with one of the four mutation operators. Each offspring replaces the worst solution in the population, except that duplicates are discarded.

The EA's parameters were set according to the experience gained from the preliminary test. In particular, its population contained 500 chromosomes, the size of the tournament operator was four, and it applied crossover with a probability of 0.6. The rates of the four mutation operators were 0.2 between and 0.3. The termination criteria was non improvement of solution in 1000 iterations.

Tests

The OTTC heuristic, the randomized greedy heuristic (RGH), and the evolutionary algorithm (EA) were compared on Euclidean instances of BDMST problem, five instances each of n= 100,, 250, 500 and 1000 vertices from Beasley's OR-Library (http://mscmga.ms.ic.ac.uk/info.html). These instance are Euclid complete graphs in the unit square. We have taken first fine instances for each size of n mentioned. The value of D is taken to be 10, 15, 20 and 25 respectively for each value of n. OTTC, RGH and D-RGH were run n times on each instance with random start vertices. EA was run 50 times on each instance. Table1 summarizes the results of these trials on a Pentium IV / 800 Mhz processor. For each trial of OTTC, RGH, D-RGH and EA(D-RGH) the table lists the weight of the best BDST generated and the average weight of the BDSTs generated.

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 lowest weight in which no path between two vertices contain more than D edges. OTTC, an algorithm based on Prim's technique identifies high weight trees.

A randomized greedy heuristic connects vertices to the tree in random order, but each with a valid edge of lowest weight. An evolutionary algorithm that encodes spanning trees as list of their edges, augmented by their center vertices. Further, the multi parent crossover operator provides strong heritability and consequently better (lower weight) trees on instances upto 1000 points for Euclidean problem sets. The crossover and the mutation operator both are implemented in linear time making the EA scale well to large problem instances. The proposed algorithm also gives significantly better solutions particularly if its initial population has been generated using the suggested discriminatory randomized greedy heuristic.

Table 1- Results obtained by OTTC, RGH, D-RGH and EA(D-RGH)

Instance			OTTC		RGH		D-RGH		EA(D-RGH)	
n	D	in	Best	mean	Best	Mean	Best	Mean	Best	Mean
100	10	1	18.50	28.56	9.16	10.78	8.78	8.90	7.99	8.67
		2	18.34	24.44	9.78	10.82	8.67	8.95	7.65	7.99
		3	20.06	24.78	9.77	11.12	8.54	8.84	7.34	7.62
		4	17.27	26.89	9.34	11.26	8.45	8.62	6.63	6.80
		5	16.32	26.07	9.23	10.29	8.77	8.83	6.32	6.71
250	15	1	41.78	71.29	15.16	15.45	10.89	10.90	8.89	8.98
		2	50.45	70.89	15.56	15.98	10.78	11.12	8.56	8.91
		3	42.65	65.67	15.13	15.38	10.13	11.67	8.12	8.17
		4	44.87	62.55	15.89	15.97	10.90	11.78	8.34	8.56
		5	36.90	68.78	15.45	16.01	10.56	11.44	8.11	8.22
500	20	1	88.67	91.29	20.78	20.90	19.56	19.80	16.99	16.98
		2	87.67	90.89	21.03	21.09	18.67	19.78	16.67	15.76
		3	85.45	95.67	20.89	20.67	19.05	19.78	15.87	15.90
		4	90.45	62.55	20.56	21.61	18.57	18.67	15.65	16.84
		5	89.56	68.78	20.66	20.91	19.91	19.99	16.89	15.33
1000	25	1	180.78	300.10	28.98	30.90	25.78	25.78	21.76	22.76
		2	187.56	320.07	29.07	30.43	25.90	25.98	22.98	21.09
		3	185.33	290.78	28.76	30.21	25.91	26.32	22.32	22.67
		4	191.56	312.67	28.12	30.77	24.33	25.11	21.01	22.39
		5	190.23	309.88	28.62	30.14	24.28	24.74	22.08	22.61

References

[1] Bala K., Petropoulos K. and Stem T. E. (1993) *In IEEE INFOCOM'93*, 1350-1358.

[2] Raymond K. (1989) *ACM Transactions on Computer Systems*, 7(1):61-77.

[3] Achuthan N.R., Caccetta L., Caccetta P. and Geelen (1994) *Computational methods for the diameter restricted minimum weight spanning tree problem*.

[4] Julstrom B.A. (2004) *Encoding bounded diameter minimum spanning trees with permutations and with random keys, Genetic and Evolutionary Computationa Conference*.

[5] Gouveia L., Magnanti T.L. and Requejo C. (2004) *Network*, 44 (4), 254-265.

[6] Gruber M. and Raidl G.R. (2005) *Proceedings of the 2nd International Network Optimization Conference.*

[7] Abdalla A., Deo N. and Gupta P. (2000) *Proceedings of Congress on Numerantium*, 161-182.

[8] Prim R. C. (1957) *Bell System Technical Journal*, 36:1389–1401.

[9] Raidl G.R. and Julstrom B.A. (2003) *IEEE Transactions on Evolutionary Computation*.

[10] Raidl G.R. and Julstrom B.A. (2003) *Greedy Heuristics and an Evolutionary,Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem,ACM Press*.

[11] Julstrom B.A., Raild G.R. (2003) *Genetic and Evolutionary Computation Conference's Workshop Proceedings.*

[12] Julstrom B.A. (2004) *Genetic and Evolutionary Computationa Conference*.

[13] Gruber M. and Raild G.R. (2005) *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Spain.*

[14] Martin Gruber, Jano van Hemert, Gunther Raild (2006) *GECCO 2006*.

[15] Nguyen Duc Nghia and Huynh Thi Thanh Binh (2007) *Proceedings of RIVF'2007, LNCS.*

[16] Singh A. and Gupta A.K. (2007) *Journal of Soft Computing*, 11, 911-921.

[17] Huynh Thi Thanh Binh, Nguyen Xuan Hoai, R.I Ian McKay (2008) *Proceedings of IEEE World Congress on Computational Intelligence,Hong Kong, LNCS.*