International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

Neighborhood Search for the Bounded Diameter Minimum Spaning Tree

Sakshi Arora[1], M. L. Garg[2]
[1,2]School of Computer Science and Engineering, Shri Mata Vaishno Devi University, Katra (J&K), India.
[1]sakshi@smvdu.ac.in, [2]garg.ml@smvdu.ac.in

Abstract

Many optimization problems including the network design problem of finding the bounded diameter minimum spanning tree are computationally intractable. Therefore, a practical approach for solving such problems is to employ heuristic algorithms that can find solution close to the optimal one within a reasonable amount of time. Neighborhood search algorithms are a wide class of algorithms where at each iteration an improved solution is found by searching the "neighborhood" of the current solution. A critical issue in the design of a neighborhood search algorithm is the choice of the neighborhood structure. Literature reveals that the larger the neighborhood, the better is the quality of the locally optimal solutions and the greater is the accuracy of the final solution obtained. At the same time, the larger the neighborhood search, the longer it takes to search the neighborhood for optimum. For this reason, an efficient search strategy is required to produce an effective heuristic in large neighborhoods. This paper focus on two known neighborhood structures for the BDMST problem. Both types of neighborhoods are large in the sense that they contain exponentially large number of candidate solutions. A novel intelligent neighborhood search technique (INST) is introduced and compared with the previously published local search techniques.

***Keywords*-- Bounded Diameter Minimum spanning tree, Local search, Heuristics.**

I. 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 $c_e \geq 0$, 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 $\geq$ 2, and whose total costs c(T) = $\Sigma_{e \in E_T} c_e$ are minimal. The BDMST problem is known to be NP-hard for 4≤D≤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). Abdalla[7] presented a greedy heuristic algorithm - the One Time Tree Construction (OTTC) for solving the BDMST problem. This algorithm is time consuming, and its performance is strongly dependent on the starting vertex. Raild and Julstrom[8] modified this approach to start from a predetermined centre and presented randomized greedy heuristic algorithm (RGH). Raidl and Julstrom proposed a genetic algorithm for solving BDMST problems which used edge set coded[9] (JR-ESEA) and permutation coded representations for individuals[10] (JRPEA). Permutation coded evolutionary algorithms were reported to give better results than edge-set coded, but usually are much more time consuming. Gruber[11] 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 neighborhood, and center change level neighborhood.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

Later, Gruber and Raild[12], re-used variable neighborhood searches, 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. The neighborhood search tends to give significantly better quality solutions if the candidate trees on which they are applied are generated using a competitive heuristic. In this paper we propose a neighborhood method Arc_Exchange() along with the detailed Remove_Arc( ) and Add_Arc( ) methods. We also suggest a neighborhood search strategy to limit the possibly exponential neighbors of a candidate and yet give results that are competitive with the search techniques available in literature.

A critical issue in the design of a neighborhood search approach is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. This choice largely determines whether the neighborhood search will develop solutions that are highly accurate or whether they will develop solutions with very poor local optima. As a rule of thumb, the larger the neighborhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. Since one generally performs many runs of a neighborhood search algorithm with different starting points, longer execution times per iteration lead to fewer runs per unit time. For this reason a larger neighborhood does not necessarily produce a more effective heuristic unless one can search the larger neighborhood in a very efficient manner.

In this paper, we develop and analyze new neighborhood structure and search procedure for the Bounded Diameter minimum spanning tree problem. The performance of a neighborhood search algorithm critically depends upon the neighborhood structure, the manner in which we define neighbors of a feasible solution. Currently, the best other available neighborhood structures are those of Amberg[13] and Sharaiha[14]. Amberg et al.'s neighborhood structure is based on exchanging single) nodes between two subtrees. The nodes can be anywhere in the subtrees and may not always be the leaf nodes of the subtrees. The neighborhood structure due to Sharaiha et al. moves a part of a subtree, from one subtree to another subtree or to the root node.

The number of neighbors in both of these neighborhood structures is no more that n^2. Both papers report computational results of subsets tabu search methods based on their neighborhood structures. We will subsequently refer to their neighborhood structures as two-exchange neighborhood structures since a neighboring solution is obtained by changing at most two subtrees. Our multi-exchange neighborhood structures allow exponentially large number of neighbors of a solution. Therefore, an explicit enumeration of the entire neighborhood will, in most cases, be prohibitively expensive. Therefore an associated efficient search strategy is required to traverse this kind of neighborhood.

II. Existing Neighborhood Structures

A neighborhood search algorithm for the Bounded diameter minimum spanning tree problem starts with a feasible tree T. Using a neighborhood structure; it defines the neighbors of T, that is, those solutions that can be obtained from T by performing an "exchange". It then identifies a suitable neighbor, replaces T by it, and repeats this process until a suitable termination criterion is reached. The performance of a neighborhood search algorithm crucially depends on the neighborhood structure. In this section, we review two existing neighborhood structures due to Amberg and Sharaiha.

2.1 Neighborhood structure due to Amberg et al.'s

The neighborhood search method due to Amberg et al. uses a node exchange procedure that transforms one feasible solution to a neighboring solution by changing the assignment of nodes in the subtrees; such a transformation is called a move. It is a two-exchange neighborhood structure since it involves at most two subtrees to define neighboring solutions. Their procedure considers two types of moves:

Node Shift: A node shift move chooses one node and moves it from its current subtree to another subtree.

Node Exchange: A node exchange move chooses two nodes belonging to different subtrees and exchanges them. The nodes selected for shift or exchange may not always be the leaf nodes of the subtrees that they belong to.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

Further, after the shift or exchange move, the subtrees readjust themselves if necessary so that each subtree is a minimum spanning tree over the node set it spans. The procedure allows only feasible moves, that is, moves that do not violate the capacity constraints.

2.2 Neighborhood structure due to Sharaiha et al.

The neighborhood search method due to Sharaiha's et al. performs cut and paste operations. A cut and paste operation consists of cutting either the whole subtree or a part of a subtree and then connecting (pasting) it either to the source node or to some other subtree. It also may be viewed as a two-exchange neighborhood structure since it involves at most two subtrees to define neighboring solutions. If the method deletes the arc (i,j) and adds in the arc (k,1), then the cost of the cut and paste operation is C_{kl} - C_{ij}. There are at most n^2 possibilities of cut and paste operations and it takes $O(K^2)$ time to identify the cost of each such operation. Consequently, the method takes $O(n^2K^2)$ time to determine the most profitable cut and paste operation.

III. PROPOSED NEIGHBORHOOD FOR THE BDMST

The Arc_Exchange() neighborhood when applied to BDMST can be viewed as the neighborhood of a candidate solution T consisting of all feasible trees differing from T in exactly a single arc or two arcs or 'k' arcs in general. The associated move can be interpreted as disconnecting some subtree and reconnecting it at another feasible place subsequently after a series of feasibility checkings (for the diameter bound and the condition of cycle free structure). We can view this neighborhood search when applied to the given problem as: (i) adding and deleting edges sequentially, (ii) accepting in parallel multiple swaps where a swap is defined by interchanging the current status of a node in a tree. But this study evaluates only the Arc_Exchange( ) neighborhood.

1-opt Neighborhood: It is the easiest to implement and consumes the least time. This neighborhood consists of $O(n^2)$ solutions. A single neighbor can be evaluated in constant time when only considering cost differences. We ensure that the diameter constraint is not violated by predetermining for each node v ϵ V the height h(v) of the subtree rooted at it; feasible candidates for becoming new predecessor of a node v after disconnecting it are all nodes at levels less than or equal to H – h(v) -1.

Hence, the total time for examining the whole neighborhood in order to identify the best move is in $O(n^2)$. The drawback incurred is compromise on diversity. The trees differing only in one arc may not lie significantly far from each other in the search space and hence entrapment in the valleys during exploration of search space cannot be avoided for long. Therefore we focus only on 2-opt and 3-opt neighborhoods for BDMSTs in our model.

IV. INST-INTELLIGENT NEIGHBORHOOD SEARCH TECHNIQUE

The Arc Exchange neighborhood explained in previous section can be searched cost effectively only if an efficient heuristic exists for searching through it. The Remove_Arc( ) operation and the Add-Arc( ) operation both can assume different forms from a set of possibilities listed in the set SELECT_ARC. SELECT_ARC lists the type of moves that can be made to select an arc for removal from the tree and the choices that can be made to select an arc for addition to the tree. Further, in our method we borrow the choice of acceptance from simulated annealing[15]. With such an acceptance criteria, the temporary solution x^t is always accepted if $c(x^t) < c(x)$, and accepted with probability $e^{-(c(xt)-c(x))/T}$ if $c(x) < c(x^t)$. Here $T > 0$ is the current Temperature. Temperature is initialized at $T_0 > 0$ and is decreased gradually, for example by performing the update $T_{new} = \alpha . T_{old}$ at each iteration, where $0 < \alpha < 1$ is a parameter. The idea is that T is relatively high initially, thus allowing deteriorating solutions to be accepted. As the search progresses T decreases and towards the end of the search only a few or no deteriorating solutions will be accepted. If such an acceptance criteria is employed, INST can be viewed as a standard simulated annealing heuristic with a complex neighborhood definition.

The Remove_Arc () method is an important art of the INST heuristic. The most important choice when implementing the above method is the degree of destruction: if only a small number of edges is removed then the heuristic may have trouble exploring the search space as the effect of a large neighborhood is lost. If a very large number of edges are removed the INST heuristic almost degrades into iterated re-optimization. This can be time consuming or yield poor quality solutions depending on how the partial solution is repaired.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

Shaw[16] proposed to gradually increase the degree of destruction, while Ropke and Pisinger[17] choose the degree of destruction randomly in each iteration by choosing the degree from a specific range dependent on the instance size. In this study we try to suggest that the Remove_Arc( ) method must also be chosen such that the entire search space can be reached, or at least the interesting part of the search space where the global optimum is expected to be found. Therefore it cannot focus on always destroying a particular component of the solution but must make it possible to remove any arc/edge from the candidate tree and should be able to destroy every part of the solution.

INST provides freedom in choosing the Add_Arc( ) method. A first decision is whether the Add_Arc( ) method should be optimal in the sense that the best possible full solution is constructed from the partial solution, or whether it should be a heuristic assuming that one is satisfied with a good solution constructed from the partial solution. An optimal Add_Arc( ) operation will be slower than a heuristic one, but may potentially lead to high quality solutions in a few iterations.

However, from a diversification point of view, an optimal Add_Arc( ) operation may not be attractive: only improving or identical-cost solutions will be produced and it can be difficult to leave valleys in the search space unless a large part of the solution is destroyed in each iteration.

It is here that the implementer of INST heuristic provides the choice of 'k' in the K-opt neighborhood exploration.

4.1 k-opt

k-opt is the most general network neighborhood. It includes all the MSTs (Minimum Spanning Trees) of a graph that are different from the given one in at most k edges. Obviously any tree can be obtained from a given one by an m-opt move. k-opt neighborhood is widely used for the CMST and some other combinatorial optimization (CO) problems, Refer, e.g., (Fischetti et al.[18], Karapetyan and Gutin[19]; Gutin and Karapetyan[20]; Snyder and Daskin[21]). It was shown to be very efficient for the TSP and some network flow based formulations of the combinatorial optimization problems (Helsgaun[22]).

In general, $N_{k\text{-}opt}(T)$ contains all the solutions that can be obtained from T by selecting k elements in T and then replacing them with k new elements such that the feasibility of the solution is preserved. In the Arc exchange () neighborhood for BDMST, k-opt means replacing k existing edges in the solution with k new edges.

The time complexity of k-opt increases exponentially with the growth of k. In practice, only 2-opt and 3-opt are used for maximum CO problems (Helsgaun[23]; Lin[24]) with rare exceptions (Helsgaun[22]). We do not consider k-opt for $k > 3$.

4.2 2-opt

For $k = 2$ and for a fixed pair of edges (T_a, T_{a+1}), (T_b, T_{b+1}) there are only two options for every 2-opt move, i.e., to replace these edges either with (T_a, T_b) and (T_{a+1}, T_{b+1}) or with (T_{b+1}, T_{a+1}) and (T_b, T_a). However, for the symmetric case both options are identical and it takes only O(1) operations to evaluate a 2-opt move, see (1). Hence, it takes $O(m^2)$ operations to explore the whole neighborhood $N_{2\text{-}opt}(T)$ in the symmetric case.

It is worth observing that the INST heuristic typically alternates between an infeasible solution and a feasible solution: the destroy operation creates an infeasible solution which is brought back into feasible form by the repair heuristic. Alternately the Arc remove and add operations can be viewed as fix/optimize operations: the fix method (corresponding to the Remove_Arc( ) method) fixes part of the solution at its current value while the rest remains free, the optimize method (corresponding to the Add_Arc () method) attempts to improve the current solution while respecting the fixed values.

V. ALGORITHM

We consider an algorithm to explore the 2-opt neighborhood. The algorithm tries all feasible pairs of a and b with $b > a$, see Algorithm below. Note that after an improvement is applied, it is not necessary to explore the whole neighborhood again.

We use an efficient approach to avoid such repetitions. In particular, the algorithm stores a flag $p(T_i)$ for every vertex T_i. This flag shows if the edge starting from Ti was changed since the last check.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

Observe that a move of Flip(Ta,b) is redundant if both edges (T_a, T_{a+1}) and (T_b, T_{b+1}) stay unchanged since the last check of Flip(T,a,b).

Algorithm: Basic 2-opt algorithm, (Symmetric case)

Input: Tree T = (T_1, T_2, . . . , T_n) and a diameter bound D.
Initialize p(T_i) ← true for every i = 1, 2, . . . ,n.
Repeat
  Initialize optima ← true.
  Initialize p`(T_i) ← false for every i = 1, 2, . . . ,n.
  for a ← 1, 2, . . . ,n−2 **do**
    for b ← a + 2, a + 3, . . . ,min {n, a + n−2} **do**
      if p(T_a) = false and p(T_b) = false **then**
        Go to the next b.
      Δ ← c(T_a, T_b) + c(T_{a+1}, T_{b+1}) − c(T_a, T_{a+1}) − c(T_b, T_{b+1}). //c is the cost associated with the edge
      if Δ < 0 then
        Replace edges (T_a, T_{a+1}) and (T_b, T_{b+1}) in T with edges (T_a, T_b) and (T_{a+1}, T_{b+1}).
        p`(T_i) ← true for every i = a, a + 1, . . . , b.
        Set optima ← false.
        Continue to the next a.
  Swap p and p`
Until optima = true

VI. Computational Results

The instances have been taken from Beasley's OR-Library[25], which were originally propodsed for 1teh Eucledian Stenier tree problem. These instances contain coordinates of points in the unit square, and the Eucledian distances between any pair of points are the edge costs. For our experiment we used the first five instances of each size n = 100, 250 and 500. The maximum diameters were set to 10, 15 and 20 respectively. All tests were performed on a Pentium 4 2.8 Ghz PC so that the results for our model could be compared with the results for VNS provided by Gruber in [11].

Regarding the termination condition we used CPU time limits of 2000, 3000 and 4000 seconds for the 100, 250 and 500 node instances.

The following table shows the values averaged over 30 runs for each of the algorithm.

TABLE 1.
FINAL SOLUTION VALUES OF LONG-TERM RUNS ON EUCLEDIAN INSTANCES

Instance			*VNS				INST			
n	D	in	best	mean	stdev	Sec.	best	mean	stdev	Sec.
100	10	1	7.759	7.819	0.03	37.35	6.524	6.857	0.03	26.56
		2	7.852	7.891	0.03	41.52	6.782	6.882	0.04	30.99
		3	7.904	7.962	0.04	38.66	6.928	7.031	0.04	27.45
		4	7.979	8.046	0.03	34.27	6.836	7.203	0.03	26.41
		5	8.165	8.203	0.03	39.31	7.182	7.347	0.03	28.50
250	15	1	12.301	12.430	0.05	1584.31	10.451	10.552	0.06	1120.44
		2	12.024	12.171	0.06	1678.90	10.230	10.329	0.07	1263.09
		3	12.041	12.112	0.04	1309.21	10.541	10.836	0.06	1090.32
		4	12.507	12.615	0.06	1572.39	10.091	10.447	0.06	1353.51
		5	12.281	12.423	0.07	1525.39	10.382	10.503	0.07	1323.60
500	20	1	16.974	17.129	0.07	3718.54	12.047	12.387	0.07	1844.94
		2	16.879	17.052	0.07	3762.02	12.822	12.538	0.09	1972.34
		3	16.975	17.148	0.0	3849.42	12.051	12.290	0.08	1981.20
		4	16.992	17.166	0.06	3687.97	12.073	12.243	0.07	1840.27
		5	16.572	16.786	0.07	3693.13	12.581	12.638	0.09	1909.83

VII. Conclusion

One of the key benefits of the INST heuristic is that a heuristic can be quickly put together from existing components: an existing construction heuristic or exact method can be turned into an Add edge heuristic and a Remove edge method based on random selection is easy to implement. Therefore we see a potential for using simple INST heuristics for benchmark purposes when developing more sophisticated methods.

INST heuristics, in general, work well when the problem considered involves constraints on MSTs such as hop constraints or diameter constraints. Such structure seems to be well suited for the large neighborhood search. For problems that do not exhibit this structure it is difficult to predict the performance of the INST heuristic and other metaheuristics may be better suited.

It is demonstrated[25] that some combinations of both small and large neighborhoods provide the best results. This could indicate that hybrid neighborhoods may be a promising direction for future research.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal,Volume 3, Special Issue 2, January 2013)
National conference on Machine Intelligence Research and Advancement (NCMIRA, 12), INDIA.

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

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

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

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

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

[13] Amberg, A., Domschke, W., Vol, S. (1996): Capacitated minimum spanning trees: Algorithms using intelligent search. Combinatorial Optinization: Theory and Practice 1, 9-39

[14] Sharaiha, Y.M., Gendreau, M., Lapoite, G., Osman, I.H. (1997): A tabu search algorithm for the capacitated shortest spanning tree problem. Networks 29, 161-171

[15] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics, 159(2):139–171, 2000.

[16] L. Perron and P. Shaw. Parallel large neighborhood search. In Proceedings of RenPar'15, 2003.

[17] S. Ropke. Parallel large neighborhood search - a software framework. In MIC 2009. The VIII Metaheuristics International Conference, 2009.

[18] Fischetti, M., Salazar Gonz´alez, J.J., Toth, P., 1997. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research 45, 378–394.

[19] Karapetyan, D., Gutin, G., 2011b. Local search heuristics for the multidimensional assignment problem. Journal of Heuristics 17, 201–249.

[20] Gutin, G., Karapetyan, D., 2010. A memetic algorithm for the generalized traveling salesman problem. Natural Computing 9, 47–60.

[21] Snyder, L., Daskin, M., 2006. A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational Research 174, 38–53.

[22] Helsgaun, K., 2009. General k-opt submoves for the Lin-Kernighan TSP heuristic. Mathematics and statistics 1,119–163.

[23] Helsgaun, K., 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research 126, 106–130.

[24] Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell System Technical Journal 45, 2245–2269.

[25] G. Gutin and D. Karapetyan. Local search heuristics for the multidimensional assignment problem. In Proc. Golumbic Festschrift, volume 5420, pages 100–115. 2009.