International Journal of Operations Research

International Journal of Operations Research Vol. 10, No. 4, 153−160 (2013)

Variable Neighborhood Search Approach for the Minimum Routing Cost Spanning Tree Problem

S. Sattari[1] and F. Didehvar[2,*]

[1,2]Department of Mathematics and Computer Science, Amirkabir University of Technology,

424 Hafez Ave, Tehran, Iran

Received October 2013; *Revised December 2013*; *Accepted December 2013*

Abstract—Given an undirected, weighted, and connected graph, in any spanning tree of such a graph, routing cost is defined as total length of all paths between each pair of vertices. The objective of minimum routing cost spanning tree is to find a spanning tree with a minimum cost. We present a basic and a modified version of the variable neighborhood search algorithm for this problem. Testing these algorithms on a set of benchmark graphs shows performance of proposed methods.

Keywords—Graph, spanning tree, routing cost, variable neighborhood search, heuristics.

1. INTRODUCTION

Consider a graph $G(V, E)$ which is undirected, weighted, and connected. In a spanning tree T of G, the routing cost between two vertices is their path length, and the routing cost of T is defined as the sum of routing costs of all pairs of vertices. The minimum routing cost spanning tree (MRCST) problem requires a spanning tree in which its routing cost is the minimum. If routing cost of two vertices u and v is shown with $d_T(u, v)$, routing cost of T will be $C(T) = \sum d_T(u,v)$. In this paper $|V|$ is shown with n; it is apparent that every solution of MRCST problem has n-1 edges.

The MRCST problem is a special case of optimal communication spanning tree problem (Hu, 1974). In optimal communication spanning tree (OCST) problem, there is a requirement for each pair of vertices that is multiplied to their path length to obtain their communication cost. When all requirements are equal to one, it is identical to the minimum routing cost spanning tree problem. The MRCST problem is NP-hard (Johnson *et al.*, 1978) and it has an application in the multiple sequence alignment problem of computational biology (Wu *et al.*, 1999). In addition, it is shown to be useful in bridged computer networks (Campos and Ricardo, 2008), and designing peer-to-peer networks (Merz and Wolf, 2006).

To find solutions of NP-hard problems, one approach is to use a metaheuristic method. Variable neighborhood search (VNS) method (Mladenović and Hansen, 1997) is a recent metaheuristic method that has been pretty successful in many optimization problems. In this study, a simple and a modified VNS algorithm are presented for the MRCST problem, and effectiveness of both algorithms with experimental results is also shown. In the rest, related studies on this problem are introduced in section 2. In section 3, an overview of VNS method is given, in section 4 our VNS algorithms are presented, and in section 5 experimental results are described. Finally, conclusions are drawn.

2. RELATED WORKS

Ahuja and Murty (1987) presented a best improvement heuristic for the OCST problem, which is also applicable to MRCST problem. In this heuristic, in each step, all edges of spanning tree are examined for removal from it. Removing each edge splits the tree into two components. All the edges that can connect these components are tested for insertion into the tree, and the resulting costs are computed. A pair of edges that removing the first one and inserting the second one results in the lowest cost tree is chosen. If this cost is lower than the cost of current tree, this change is applied to the current tree and algorithm continues, otherwise, it stops.

Wong (1980) proposed a 2-approximation algorithm, in which for each vertex, a single source shortest paths algorithm is executed; resulting in a shortest paths tree. Of these trees, the tree in which the total routing cost from the root to other vertices is the least is selected as the output of the algorithm.

Wu *et al.* (1999) proved that MRCST problem is reducible to metric case. They also proposed a polynomial approximation scheme. In another study, Wu *et al.* (2000) presented approximation algorithms with factors 2, 15/8, and 3/2 having time complexities $O(n^2)$, $O(n^3)$, and $O(n^4)$, respectively. Additionally, they showed that for each positive constant ε, the $(4/3+\varepsilon)$ approximation is reachable within polynomial time.

Fischetti *et al.* (2002) used branch and price method for MRCST problem. In their approach for constructing initial solutions, a local search algorithm was proposed. In each iteration, this local search algorithm tries all edges outside the current

[*] Corresponding author's email: didehvar@aut.ac.ir

tree for insertion into it. Insertion of an edge creates a cycle that is broken by deleting another edge of the tree. The best tree that can be constructed within this way becomes the next tree if its cost is less than the current tree; otherwise, the algorithm ends.

Grout (2005) presented a heuristic algorithm for the connected vertex cover problem that was discussed as a solution of MRCST problem in some papers. This heuristic ignores edge weights and is suitable just for homogenous graphs.

Campos and Ricardo (2008) by combining ideas of Wong (1980) and Grout (2005) developed a greedy construction algorithm. This deterministic algorithm has some parameters whose values were found by simulation. They tested their algorithm on some random graphs and found better results compared with Wong (1980).

Julstrom (2005) presented a stochastic hill climber and two genetic algorithms. The hill climber in each step generates a random neighbor of current tree that differs from it in one edge, and moves to that if it is better than current tree. The main difference between presented genetic algorithms is in the representation method of spanning tree. One algorithm represents spanning tree with a set of edges, and the other uses blob code (Picciotto, 1999). Experimental results proved superiority of stochastic hill climber over these genetic algorithms.

Merz and Wolf (2006) introduced three heuristics and used them in evolutionary algorithms. In the first and second heuristics, parent of one node is changed; in the first one the new parent is selected from neighbors of current parent, but in the second approach, it is selected randomly. In the third heuristic, two nodes swap their parents. In their evolutionary methods, there is only mutation and no crossovers. A tree is mutated by deleting some edges and inserting new edges. They executed their algorithms on some graphs derived from measuring delays in internet.

Wolf and Merz (2010) presented another evolutionary algorithm. In their algorithm, they used the local search method of Ahuja and Murty (1987) and proposed two heuristics. In the first heuristic, a new parent is selected for a node from those nodes belonging to the same subtree as itself. In the second heuristic, parents of two nodes are exchanged with each other. They tested three versions of their evolutionary algorithm on the dataset used in Merz and Wolf (2006).

Singh (2008) presented a perturbation based local search algorithm. It constructs initial solution by a method like Prim's algorithm (1957). It adds edges to the tree one by one, but probability of adding an edge is inversely related to its weight. Then, it starts an iterative method, and applies a local search method to the current solution in each of iterations. Whenever algorithm finds a better solution, it replaces the current solution. After a number of non-improving iterations, the current solution is perturbed by deleting and adding some edges. The local search method of this algorithm randomly deletes one edge of tree and adds another edge to the tree that leads to the lowest possible cost. The added edge in one iteration is prohibited from deletion in the next iteration. They executed their algorithm on the dataset introduced in Julstrom (2005) and found better results.

Singh and Sundar (2011) proposed an artificial bee colony (ABC) algorithm for MRCST problem. To construct an initial solution, at first, they insert one random vertex into tree. Then, in each step probability of adding an edge is inversely related to its weight or the square of its weight; this is decided randomly for the entire tree in the beginning. The most important idea of their population based method is generating neighbors of a tree by randomly deleting an edge of current solution and adding another edge that exists in a different solution. In their improved algorithm, they applied local search of Ahuja and Murty (1987) to the result of ABC algorithm. Experiments were done on benchmark data set of Julstrom (2005) and its results were better than Julstrom (2005) and Singh (2008).

3. VARIABLE NEIGHBORHOOD SEARCH METHOD

VNS method is a simple metaheuristic algorithm based on local search algorithms and defining neighborhoods for solutions. The basic idea of this algorithm is that for every problem, the global optimum is a local optimum in some neighborhoods, so it searches different neighborhoods to find better solutions. In this algorithm, for each solution of the given problem, a set of neighborhoods N_k ($k = 1, ..., L$) should be defined. Algorithm starts with an initial solution. In each iteration, first, it randomly generates a solution in current neighborhood N_k. Then, it applies local search to the solution. If the resulted solution is better than the current one, it becomes the current solution and the algorithm returns to search in N_1, otherwise, algorithm searches in next neighborhood i.e. N_{k+1}. Additionally, if current neighborhood is the biggest one, i.e. $k = L$, it returns to N_1. Algorithm 1 gives the outline of simple VNS algorithm.

VNS was introduced by Mladenović and Hansen (1997) and has been applied on many problems like degree-constrained minimum spanning tree problem (Ribeiro and Souza, 2002), maximum clique problem (Hansen *et al.*, 2004), and graph coloring (Avanthay *et al.*, 2003).

Algorithm 1: SimpleVNS()
Output: s, a solution of problem

```
1: $s$ = an initial solution of problem
2: $k$ =1
3: while stopping criteria not met do
4:     $s_1$ = a random solution in neighborhood $N_k$ of $s$
5:     $s_2$ = LocalSearch ($s_1$)
6:     if $s_2$ is better than $s$ then
7:         $s = s_2$
```

```
 8:         $k = 1$
 9:     **else**
10:         $k = k + 1$
11:         **if** $k > L$ **then** $k = 1$
12:     **end if**
13: **end while**
14: **return** $s$
```

4. VARIABLE NEIGHBORHOOD SEARCH METHOD

To design a VNS algorithm, we specify three components: a method to generate an initial solution, neighborhood structure, and a local search algorithm. In our VNS algorithms, we use local search algorithm of Ahuja and Murty (1987), which is a best improvement algorithm. It deletes an edge and adds another edge to current tree in each step that results in the lowest possible routing cost, and this continues so that no change can generate a better tree than the current one. This algorithm is given in the appendix. In the following, we describe our algorithms for generating initial solutions, and generating neighbors. Then we present a VNS algorithm for the MRCST problem. Moreover, we propose a modified VNS algorithm at the end of the section.

4.1 Initial solution

To generate an initial solution, we use a method similar to Wong (1980). We generate all shortest paths trees, but instead of just computing routing cost from the root of tree to other vertices, we compute total routing cost of the tree. We select a tree with the least routing cost as the result of this algorithm. Algorithm 2 shows our algorithm for generating initial solution.

Algorithm 2: InitialSolution(G)
Input: Graph $G(V, E)$
Output: S, the least cost shortest paths tree

```
1: $c = \infty$
2: **for** $i = 1$ to $n$ **do**
3:     $T$ = shortest paths tree rooted at $v_i$
4:     **if** $C(T) < c$ **then**
5:         $S = T$
6:         $c = C(T)$
7:     **end if**
8: **end for**
9: **return** $S$
```

4.2 Neighborhood structure

For every number k from 1 to n-2, we define a neighborhood N_k for any solution of the problem. For a given spanning tree this neighborhood is defined as the set of spanning trees that differ from the original tree in k edges. The maximum value for k is n-2 because we assume that a tree and its neighbor share at least one edge. To generate a random member of N_k, we randomly select k edges and delete them from the tree one by one. After deleting each edge, we add another edge to the tree that connects two appeared components. To choose this edge, first, we select a vertex from one component randomly, then of all the edges that connect this vertex to the other component, we choose one whose weight is the minimum and add it to the tree. Neighbor generation method is given in algorithm 3.

Algorithm 3: GenerateNeighbor(G, T, k)
Input: Graph $G(V, E)$. Tree T with edges $e_1, e_2, ..., e_{n-1}$. A number $k \in \{1, 2, ..., n-2\}$
Output: S, a tree with k different edges from T

```
1: $S = T$
2: Randomly generate distinct numbers $r_1, r_2, ..., r_k$ such that $\forall i \in \{1, 2, ..., k\}$: $r_i \in \{1, 2, ..., n-2\}$
3: **for** $i = 1$ to $k$ **do**
4:     Delete edge $e_{r_i}$ from $S$, Assume this generates two components $A$, $B$
5:     Select one random vertex $v \in A$
6:     Select a vertex $w \in B$ such that $\forall\ u \in B$: weight($v$, $w$) $\le$ weight($v$, $u$)
7:     Add edge ($v$,$w$) to $S$
8: **end for**
9: **return** $S$
```

4.3 VNS algorithm

We described required components of a VNS algorithm; using them, we present our simple VNS approach for MRCST problem in algorithm 4.

```
Algorithm 4: VNS-MRCST($G$)
Input: Graph $G(V, E)$
Output: $S$, a spanning tree of $G$
 1: $S$ = InitialSolution($G$)
 2: $i = 1$
 3: while stopping criteria not met do
 4:     $S_1$ = GenerateNeighbor($G$, $S$, $i$)
 5:     $S_2$ = LocalSearch($G$, $S_1$)
 6:     if $C(S_2) < C(S)$ then
 7:         $S = S_2$
 8:         $i = 1$
 9:     else
10:         $i = i + 1$
11:         if $i > n - 2$ then $i = 1$
12:     end if
13: end while
14: return $S$
```

4.4 Modified VNS algorithm

In the basic VNS algorithm, after finding a better solution, algorithm returns to the neighborhood N_1. However, we introduce the idea that if current neighborhood is capable of finding a good solution, possibly, we can continue to search for better solutions in this neighborhood. The only difference between this algorithm and the previous one is removing line 8 of the algorithm. Algorithm 5 implements this idea.

```
Algorithm 5: Modified-VNS-MRCST($G$)
Input: graph $G(V, E)$
Output: $S$, a spanning tree of $G$
 1: $S$ = InitialSolution($G$)
 2: $i = 1$
 3: while stopping criteria not met do
 4:     $S_1$ = GenerateNeighbor($G$, $S$, $i$)
 5:     $S_2$ = LocalSearch($S_1$)
 6:     if $C(S_2) < C(S)$ then
 7:         $S = S_2$
 8:     else
 9:         $i = i + 1$
10:         if $i > n - 2$ then $i = 1$
11:     end if
12: end while
13: return $S$
```

5. EXPERIMENTAL RESULTS

To test our algorithms experimentally, we implemented and executed them on a set of benchmark problems. We implemented our VNS algorithms using C language and executed them on a Pentium 4 Windows XP machine with a 2.8 GHz CPU and 512 MB RAM.

We used 35 benchmark instances that were proposed by Julstrom (2005). These problems are categorized into two groups. The first group contains 21 problems that were primarily designed for Euclidean Steiner tree problem and are available from OR-Library (Beasley, 2005). Actually, these instances consist of points located in a unit square of plane that we consider them as vertices of a complete graph. There are seven graphs with 50 vertices, seven with 100 vertices, and seven with 250 vertices. The second group consists of 14 problems that were generated randomly by Julstrom (2005) especially for this problem. This set includes complete graphs having 100 and 300 vertices; for each size of which there are seven instances. In these graphs, edge weights were chosen from interval [0.01, 0.99].

These benchmark problems have been studied by some papers with which we compare our algorithms. These algorithms are compared with perturbation based local search (PB-LS) algorithm of Singh (2008) and artificial bee colony with local search

(ABC+LS) algorithm of Singh and Sundar (2011). In this section, we present algorithm 4 with the name of VNS and use MVNS for algorithm 5.

For each graph, we executed our algorithms 30 times. We stopped running our algorithms when we reached the best known value for each instance, or when we spent a specified time. We derived best known values from Singh and Sundar (2011). Considering average execution times found in Singh and Sundar (2011) and features of our machine, we set maximum 40 second for graphs of size 50 and 100, and allocated maximum 300 seconds for graphs having 250 or 300 vertices.

Table 1 shows best values and mean values of routing costs found by these algorithms on Euclidean benchmark instances. In each row of this table, better values are shown in bold. We see that in size 50 graphs best solutions of all algorithms are the same. However, in graphs that have 100 vertices, only PB-LS results are not as well as other algorithms. In graphs of size 250, best values found by VNS are always better than algorithm PB-LS, and they are also better than ABC+LS, except for e250.7 instance. Best solutions of MVNS are better than PB-LS and ABC+LS. Moreover, best values of MVNS are better than VNS except for instance e250.6.

Considering mean values, our two algorithms outperform other algorithms; only in four graphs of size 50, one of the algorithms PB-LS and ABC+LS have found the best mean value. Mean values of MVNS are the best in graphs of size 250. Only in graphs e100.6 and e100.7, average values found by MVNS are worse than VNS.

Table 1. Best and mean values found by PB-LS, ABC+LS, VNS, and MVNS on Euclidean graphs

Instance	**PB-LS**		**ABC+LS**		**VNS**		**MVNS**	
	Best	**Mean**	**Best**	**Mean**	**Best**	**Mean**	**Best**	**Mean**
e50.1	**983.5**	983.6	**983.5**	983.6	**983.5**	**983.5**	**983.5**	**983.5**
e50.2	**901.3**	901.5	**901.3**	**901.3**	**901.3**	**901.3**	**901.3**	**901.3**
e50.3	**888.3**	888.9	**888.3**	888.7	**888.3**	**888.3**	**888.3**	**888.3**
e50.4	**776.9**	777.0	**776.9**	**776.9**	**776.9**	**776.9**	**776.9**	**776.9**
e50.5	**847.9**	**847.9**	**847.9**	848.0	**847.9**	**847.9**	**847.9**	**847.9**
e50.6	**818.1**	**818.1**	**818.1**	818.2	**818.1**	**818.1**	**818.1**	**818.1**
e50.7	**865.6**	865.7	**865.6**	866.1	**865.6**	**865.6**	**865.6**	**865.6**
e100.1	**3507.0**	3510.4	**3507.0**	3507.9	**3507.0**	**3507.6**	**3507.0**	**3507.6**
e100.2	3308.0	3310.0	**3307.9**	3308.7	**3307.9**	**3307.9**	**3307.9**	**3307.9**
e100.3	**3566.3**	3567.7	**3566.3**	3566.5	**3566.3**	**3566.3**	**3566.3**	**3566.3**
e100.4	3448.2	3451.3	**3448.1**	3451.0	**3448.1**	3448.5	**3448.1**	**3448.4**
e100.5	3637.7	3643.3	**3637.0**	3639.4	**3637.0**	**3637.1**	**3637.0**	**3637.1**
e100.6	3437.6	3442.0	**3436.5**	3438.0	**3436.5**	**3437.8**	**3436.5**	3438.0
e100.7	3703.7	3706.4	**3703.5**	3704.6	**3703.5**	**3703.5**	**3703.5**	3703.7
e250.1	22137.4	22199.2	22089.6	22144.8	22080.9	22148.4	**22067.2**	**22098.7**
e250.2	22797.9	22970.8	22775.2	22838.6	**22768.7**	22774.8	**22768.7**	**22772.4**
e250.3	21888.8	22069.4	21886.1	21927.7	21879.2	21933.7	**21869.1**	**21884.6**
e250.4	23456.6	23581.4	23428.5	23467.6	23420.2	23439.8	**23417.6**	**23425.6**
e250.5	22420.1	22492.9	22386.9	22423.8	22380.5	22407.8	**22378.4**	**22391.2**
e250.6	22312.6	22397.2	22285.3	22314.2	**22284.0**	22302.7	22284.1	**22299.2**
e250.7	22936.5	23003.3	22923.9	22966.2	22926.8	22996.2	**22908.2**	**22938.2**

In Table 2 for each random instance and each algorithm best values and mean values of routing costs are presented. On these graphs, best values of these algorithms are equal in all of the instances. Mean values found values by VNS and MVNS are the same. Mean values of ABC+LS are equal to the mean values of VNS and MVNS, with the exception of graph r300.6 in which it is a bit worse. Mean values of PB-LS are weaker than the other algorithms in graphs of size 300.

Table 2. Best and mean values found by PB-LS, ABC+LS, VNS, and MVNS on random graphs

Instance	**PB-LS**		**ABC+LS**		**VNS**		**MVNS**	
	Best	**Mean**	**Best**	**Mean**	**Best**	**Mean**	**Best**	**Mean**
r100.1	**597.9**	**597.9**	**597.9**	**597.9**	**597.9**	**597.9**	**597.9**	**597.9**
r100.2	**586.0**	**586.0**	**586.0**	**586.0**	**586.0**	**586.0**	**586.0**	**586.0**
r100.3	**607.0**	**607.0**	**607.0**	**607.0**	**607.0**	**607.0**	**607.0**	**607.0**
r100.4	**598.4**	**602.1**	**598.4**	**598.4**	**598.4**	**598.4**	**598.4**	**598.4**
r100.5	**624.4**	**624.4**	**624.4**	**624.4**	**624.4**	**624.4**	**624.4**	**624.4**
r100.6	**615.5**	**615.5**	**615.5**	**615.5**	**615.5**	**615.5**	**615.5**	**615.5**
r100.7	**514.7**	**514.7**	**514.7**	**514.7**	**514.7**	**514.7**	**514.7**	**514.7**
r300.1	**4131.1**	4196.1	**4131.1**	**4131.1**	**4131.1**	**4131.1**	**4131.1**	**4131.1**
r300.2	**4040.7**	4131.2	**4040.7**	**4040.7**	**4040.7**	**4040.7**	**4040.7**	**4040.7**
r300.3	**4134.8**	4220.6	**4134.8**	**4134.8**	**4134.8**	**4134.8**	**4134.8**	**4134.8**
r300.4	**4229.3**	4272.6	**4229.3**	**4229.3**	**4229.3**	**4229.3**	**4229.3**	**4229.3**
r300.5	**3951.9**	4041.6	**3951.9**	**3951.9**	**3951.9**	**3951.9**	**3951.9**	**3951.9**
r300.6	**4314.4**	4458.8	**4314.4**	4314.5	**4314.4**	**4314.4**	**4314.4**	**4314.4**
r300.7	**4093.9**	4299.4	**4093.9**	**4093.9**	**4093.9**	**4093.9**	**4093.9**	**4093.9**

If we compare tables 1 and 2, we see that algorithms VNS and MVNS have found new results for Euclidean instances, but their results on random instances are similar to the other algorithms. This may indicate that our algorithms were not able to improve best and mean values for random instances. But we believe that the values on random instances are optimal. Even executing our algorithms for longer times didn't yield better values. Our results and results of the other algorithms show that these random instances are much easier to solve compared to the Euclidean instances.

In tables 3 and 4 we give standard deviation of results and average execution times of these algorithms in 30 runs. In both tables for VNS and MVNS algorithms if on an instance the value of standard deviation is zero it means that the algorithm has found the same value of routing cost in all of the runs on that instance. In addition, when we have obtained a value that is very small but not zero, we have indicated it by "< 0.1" notation.

Average execution times of algorithms PB-LS and ABC+LS have been published to one decimal place. But we reached some average execution times that rounding them to one decimal place wouldn't show their real value; so we give our results for VNS and MVNS algorithms to two decimal places.

Table 3 reports standard deviation and average execution times for Euclidean benchmark graphs. Standard deviations of our algorithms are less than PB-LS and ABC+LS; these algorithms have reached best standard deviation only in 2 and 4 graphs, respectively. VNS is better than MVNS in 4 graphs and MVNS is better than it in 6 graphs.

Execution environment of PB-LS and ABC+LS was a 3.0 GHz Pentium 4 under Red Hat Linux 9.0, which is different from our machine. Additionally, PB-LS executes a loop for 50000 times and ABC+LS stops its main algorithm after 20*n* non-improving iterations, and then runs a local search algorithm. These termination conditions differ from ours. Therefore, we cannot give an exact comparison between our approaches and the other algorithms. However, we can see that on Euclidean instances ABC+LS is always faster than PB-LS. Algorithm MVNS is faster than VNS in 14 graphs especially in graphs of size 250, but it is slower than VNS in 7 graphs.

Table 4 shows standard deviation of results and average execution times of these algorithms in 30 runs for random benchmark instances. For random graphs, ABC+LS, VNS, and MVNS have similar standard deviations, however, PB-LS algorithm is weaker. MVNS is faster than VNS on 2 instances, and they have reached the same average time on 8 instances.

Table 3. Standard deviation and average execution times of PB-LS, ABC+LS, VNS, and MVNS on Euclidean graphs

Instance	**PB-LS**		**ABC+LS**		**VNS**		**MVNS**	
	SD	**Time**	**SD**	**Time**	**SD**	**Time**	**SD**	**Time**
e50.1	0.2	7.5	0.1	4.7	0	0.33	0	0.34
e50.2	0.1	7.9	0.0	3.6	0	0.85	0	0.38
e50.3	0.1	7.6	0.4	4.4	0	0.14	0	0.19
e50.4	0.4	8.2	0.0	3.2	0	0.35	0	0.24
e50.5	0.0	8.7	0.0	3.2	0	0.35	0	0.40
e50.6	0.0	8.2	0.0	4.1	0	0.31	0	0.17
e50.7	0.2	8.0	0.5	4.1	< 0.1	0.50	< 0.1	0.62
e100.1	2.7	54.7	1.1	29.7	0.8	30.83	0.9	30.04
e100.2	2.0	53.9	1.1	28.9	< 0.1	5.66	< 0.1	6.31
e100.3	0.9	60.6	0.8	25.1	0	3.45	< 0.1	2.73
e100.4	4.4	53.5	2.2	25.1	0.2	36.28	0.2	33.89
e100.5	6.4	50.5	1.9	29.5	0.2	23.15	0.2	20.63
e100.6	2.3	56.5	2.7	28.9	1.5	30.22	1.5	31.68
e100.7	2.6	55.4	2.0	28.5	< 0.1	10.37	0.7	10.91
e250.1	77.1	590.5	33.0	266.9	42.3	290.45	24.2	248.05
e250.2	117.4	573.3	54.6	277.2	12.1	88.78	2.5	53.78
e250.3	161.2	590.7	16.5	303.0	74.7	266.59	9.1	163.51
e250.4	101.3	573.0	19.9	309.5	33.7	271.04	5.7	150.69
e250.5	50.7	563.3	30.0	296.5	17.6	281.85	7.1	224.62
e250.6	86.6	605.0	22.3	377.0	5.7	291.48	6.4	282.83
e250.7	34.6	585.4	30.4	321.0	31.9	296.90	19.7	251.83

Table 4. Standard deviation and Average execution times of PB-LS, ABC+LS, VNS, and MVNS on random graphs

Instance	**PB-LS**		**ABC+LS**		**VNS**		**MVNS**	
	SD	**Time**	**SD**	**Time**	**SD**	**Time**	**SD**	**Time**
r100.1	0.0	52.9	0.0	16.3	0	0.04	0	0.04
r100.2	0.0	54.5	0.0	16.3	0	0.04	0	0.04
r100.3	0.0	52.6	0.0	11.1	0	0.18	0	0.20
r100.4	3.0	51.6	0.0	16.5	< 0.1	2.49	< 0.1	1.94
r100.5	0.0	54.9	0.0	16.5	0	0.03	0	0.03
r100.6	0.0	54.4	0.0	16	0	0.04	0	0.04
r100.7	0.0	53.1	0.0	14.8	0	0.03	0	0.03
r300.1	91.3	709.1	0.0	472.4	0	1.03	0	1.03
r300.2	138.2	674.3	0.0	307.9	< 0.1	165.41	< 0.1	90.37
r300.3	82.5	697.1	0.0	467.8	0	0.80	0	0.81
r300.4	31.6	668.2	0.0	364.7	0	0.90	0	0.90
r300.5	69.8	681.1	0.0	272.6	0	0.88	0	0.89
r300.6	52.9	681.3	0.0	600	0	0.82	0	0.82
r300.7	159.7	689.4	0.0	383.5	0	0.64	0	0.65

Average time values in table 4 for VNS and MVNS algorithms are much smaller in comparison with average time values in table 3. This happened because solutions of these random benchmark instances are easier to find. Moreover the method we have used for our initial solution generation finds solutions that are quite close to optimal solutions. Therefore our algorithms on random instances reach the best value in smaller steps. Additionally, in table 4, difference between average execution times of our algorithms and the two other algorithms are quite large. This can be explained by the fact that VNS and MVNS algorithms unlike other algorithms stop after finding the best known value, and on random instances they find it very soon.

6. CONCLUSION

We presented a simple variable neighborhood search algorithm for minimum routing cost spanning tree problem. We also proposed a modified variable neighborhood search algorithm. Results of running our algorithms on a set of benchmark problems showed that these two algorithms can find new best values in some of the benchmark instances. Our algorithms are superior to the best known metaheuristic algorithms of this problem, especially as they find high quality solutions in average, and have lower standard deviation.

ACKNOWLEDGEMENTS

We would like to thank Prof. Julstrom for providing the random dataset. Also, we gratefully thank anonymous referees for their valuable comments.

REFERENCES

1. Ahuja, R.K. and Murty, V.V.S. (1987). Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Science*, 21(3): 163-170.
2. Avanthay, C., Hertz, A. and Zufferey, N. (2003). A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2): 379-388.
3. Beasley, J. (2005). OR-Library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html
4. Campos, R. and Ricardo, M. (2008). A fast algorithm for computing minimum routing cost spanning trees. *Computer Networks*, 52(17): 3229-3247.
5. Fischetti, M., Lancia, G. and Serafini, P. (2002). Exact algorithms for minimum routing cost trees. *Networks*, 39(3): 161-173.
6. Johnson, D.S., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1978). The complexity of the network design problem. *Networks*, 8(4): 279-285.
7. Grout, V. (2005). Principles of cost minimisation in wireless networks. *Journal of Heuristics*, 11(2): 115-133.
8. Hansen, P., Mladenović, N. and Urošević, D. (2004). Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145(1): 117-125.
9. Hu, T.C. (1974). Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3): 188-195.
10. Julstrom, B.A. (2005). The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. *Proceeding of 2005 conference on Genetic and evolutionary computation*, ACM, pp. 585-590.
11. Merz, P. and Wolf, S. (2006) Evolutionary local search for designing peer-to-peer overlay topologies based on minimum routing cost spanning trees. In *Parallel Problem Solving from Nature-PPSN IX*, Springer Berlin Heidelberg, pp. 272-281.
12. Mladenović, N. and Hansen, P. (1997). Variable neighborhood search, *Computers & Operations Research*, 24(11): 1097-1100.
13. Picciotto, S. (1999). *How to encode a tree.* PhD thesis, San Diego, University of California.
14. Prim, R.C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6): 1389-1401.
15. Ribeiro, C.C. and Souza, M.C. (2002). Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118(1): 43-54.
16. Singh, A. (2008). A new heuristic for the minimum routing cost spanning tree problem. *Proceedings of International Conference on Information Technology (ICIT'08)*, IEEE, pp. 9-13.
17. Singh, A. and Sundar S. (2011). An artificial bee colony algorithm for the minimum routing cost spanning tree problem. *Soft Computing*, 15(12): 2489-2499.
18. Wolf S. and Merz P. (2010). Efficient cycle search for the minimum routing cost spanning tree problem. *Proceedings of 10th European Conference on Evolutionary Computation in Combinatorial Optimization*, Springer Berlin Heidelberg, pp. 276-287.
19. Wong, R.T. (1980). Worst-case analysis of network design problem heuristics. *SIAM Journal on Algebraic Discrete Methods*, 1: 51-63.
20. Wu, B.Y., Chao, K.M. and Tang, C.Y. (2000). Approximation algorithms for the shortest total path length spanning tree problem. *Discrete Applied Mathematics*, 105(1): 273-289.
21. Wu, B.Y., Lancia, G., Bafna, V., Chao, K.M., Ravi, R. and Tang, C.Y. (1999). A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3): 761-778.

APPENDIX A.

Details of local search method mentioned in section 4 and used by algorithms 4 and 5 are given in the following listing:

Algorithm 6: LocalSearch(G, T)
Input: Graph $G(V, E)$. A spanning tree T of G
Output: S, a modified spanning tree

```
1. $S = T$
1: $b = 1$
2. while $b$=1 do
3.    $c_{min} = C(S)$
4.    $b = 0$
5.    foreach edge $e \in S$ do
6.       $S = S - \{e\}$
7.       Find set $F$ of edges different from $e$ such that each of them joins components of $S$
8.       foreach edge $f \in F$ do
9.          if $C(S \cup \{f\}) < C_{min}$ then
10.            $c_{min} = C(S \cup \{f\})$
11.            $e_{del} = e$
12.            $e_{ins} = f$
13.            $b = 1$
14.         end if
15.      end for
16.      $S = S \cup \{e\}$
17.   end for
18.   if $b = 1$ then $S = S - \{e_{del}\} \cup \{e_{ins}\}$
19. end while
20. return $S$
```