A METAHEURISTIC ALGORITHM FOR THE MINIMUM ROUTING COST SPANNING TREE PROBLEM

Conference Paper · January 2013

CITATIONS
2

READS
48

2 authors, including:

Farzad Didehvar
Amirkabir University of Technology
63 PUBLICATIONS **150** CITATIONS

SEE PROFILE

Extended Abstracts of the International Conference on Operations Research and Optimization
January 19-22, 2013, University of Tehran, Tehran, Iran.

A METAHEURISTIC ALGORITHM FOR THE MINIMUM ROUTING COST SPANNING TREE PROBLEM

S. SATTARI*, F. DIDEHVAR

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE, AMIRKABIR UNIVERSITY OF TECHNOLOGY.

{S.SATTARI,DIDEHVAR}@AUT.AC.IR

Abstract. The routing cost of a spanning tree in a weighted and connected graph is defined as the total cost of paths between all pairs of vertices. Minimum routing cost spanning tree problem is an NP-Hard problem. We present a GRASP with path-relinking metaheuristic algorithm for this problem. GRASP method is a multi-start algorithm that in each iteration constructs a randomized greedy solution and applies local search to it. Path-relinking stores elite solutions and in the search of better solutions explores the paths between solutions. Experimental results show the performance of our algorithm on many benchmark problems compared to the best known algorithm.

1. Introduction

Consider an undirected connected graph $G(V, E)$ with nonnegative edge weights and spanning tree T of it. We show the path length between vertices v and u by $d_T(u, v)$ and call it routing cost of these vertices. The routing cost of the tree T is defined as sum of these costs i.e. $C(T) = \sum d_T(u, v)$. Minimum routing cost spanning tree (MRCST) is a spanning tree T such that $C(T)$ is minimum.

This is an NP-Hard problem [1]. Applications of MRCST problem are in the bridging of heterogeneous networks [2], and multiple sequence alignment problem of computational biology [3].

Key words and phrases. graph, spanning tree, routing cost, metaheuristic

* Speaker.

There are many works on this problem. Wong using shortest paths trees presented a 2-approximation algorithm [4]. Julstrom gave two genetic algorithms and a stochastic hill-climber [5]. Campos presented a greedy construction algorithm [2]. Singh proposed a perturbation based local search algorithm [6]. Singh and Sundar used an artificial bee colony (ABC) approach with local search [7].

In this paper, we present a GRASP with path-relinking metaheuristic algorithm for this problem and report its performance on a set of benchmark instances.

2. GRASP with path-relinking for MRCST

Greedy randomized adaptive search procedure (GRASP) is a multi-start metaheuristic method. In each iteration, it constructs a solution using a randomized greedy algorithm. Then runs a local search on the obtained solution. The best solution over all iterations is the output of the algorithm.

Path-relinking is memory-based method that stores elite solution in a pool P, and explores trajectories connecting different solutions in order to find other solutions. In GRASP with path-relinking method after local search phase, one elite solution from pool P is selected randomly, and the path between current solution of GRASP and this elite solution is explored.

In our algorithm, to construct a solution, first using Dijkstra's algorithm we generate all shortest paths trees, and calculate routing costs of them. We memorize roots of two least cost trees. Each time to generate a spanning tree, we randomly select one of these roots name r, and then in a way similar to Prim's algorithm add other vertices to the tree. Consider A as the set of current vertices of spanning tree, we choose a vertex w from $V - A$ such that there exists an edge (v, w), where v is a member of A. If we show weight of (v, w) with $c(v, w)$, then probability of selecting this edge is computed as follows: $\frac{c(v,w)\times d_T(r,w)}{\sum_{v\in A, w\in V-A} c(v,w)\times d_T(r,w)}$

A known heuristic for changing a given spanning tree is to remove an edge from tree to split it into two separate components and then add another edge to join them again. We use a best improvement method; we try deleting each edge of the spanning tree. Then we find all possible replacement edges and calculate resulting routing costs. If no replacement produces a tree with lower cost than current tree, local search ends. Otherwise, the best pair of edges is chosen and this replacement is applied to the tree and algorithm continues with this new spanning tree.

In each iteration, after local search phase of GRASP and finding a solution g, the path-relinking procedure starts. First, it selects a random solution p form the pool P. From g and p if we call the solution with lower cost s_1 and the other s_2, path-relinking considers s_1 as the current spanning tree and in each step removes an edge of s_1 from current tree and adds to it another edge from s_2. This process is done using best improvement strategy. Note that depending on the structures

of s_1 and s_2, the tree transformation procedure may not complete up to the end. In the trajectory explored by path-relinking, the best solution is selected and it is tested for insertion into the pool P. When the pool is not full, every new solution is inserted into it. After that when a new solution is found, if it is better than the worst solution of the pool, then the new solution replaces it in the pool.

3. Experimental results

In order to show effectiveness of the proposed approach, we implemented our algorithm using C++ language and executed it on a set of 35 benchmark instances that are proposed by Julstrom [5]. There are two groups of problems. The first group contains seven Euclidean graphs for each size of 50, 100, and 250 vertices. The second group consists of seven random graphs for each size of 100, and 300 vertices. Results of our method on these benchmark problems are compared to [7]. It has used an artificial bee colony algorithm with local search (ABC+LS) and obtained the best results on these benchmark problems.

The only parameter of our GRASP with path-relinking algorithm (GRASP+PR) is pool size that we set it to $4\sqrt{|V|}$. For this algorithm, we allocate a time limit of 40 seconds for graphs having at most 100 vertices, and 300 seconds for larger graphs. In addition, if we reach the best known value of the routing cost of the input graph, we stop the algorithm. We executed our algorithm for 30 runs for each problem.

In table 1, we report obtained results. For each problem, there are four columns: the best value found, the average value in 30 runs, the standard deviation, and the average run time in seconds. For each instance, the smallest best and mean values are shown in boldface. Running environments of these algorithms is different, so we cannot exactly compare the running times. We executed our algorithm on a 2.8 GHz Pentium 4 Windows XP machine, but algorithm ABC+LS was executed on a 3.0 GHz Pentium 4 under Red Hat Linux 9.0. However, we can see that in most of the instances GRASP+PR has less running time. Except the e250.6 instance the best and mean values found by GRASP+PR are never less than ABC+LS. In addition, it found new best values for six other Euclidean graphs of size 250, and the average values obtained by GRASP+PR algorithm are better in 19 instances.

4. Conclusions

We presented a GRASP with path-relinking algorithm for the MRCST problem. Computational results on a set of benchmark problems showed the effectiveness of our algorithm. This fast algorithm improved the best known values for some benchmark instances. In addition, compared to the best published algorithm, the average quality of solutions is better.

Table 1
Results of ABS+LS and GRASP+PR algorithms on benchmark instances

	ABC+LS				GRASP+PR			
Instance	Best	Mean	SD	Time (s)	Best	Mean	SD	Time (s)
e50.1	**983.5**	983.6	0.1	4.7	**983.5**	**983.5**	0.0	0.3
e50.2	**901.3**	**901.3**	0.0	3.6	**901.3**	**901.3**	0.0	0.5
e50.3	**888.3**	888.7	0.4	4.4	**888.3**	**888.3**	0.0	0.7
e50.4	**776.9**	**776.9**	0.0	3.2	**776.9**	**776.9**	0.0	0.6
e50.5	**847.9**	848.0	0.0	3.2	**847.9**	**847.9**	0.0	0.1
e50.6	**818.1**	818.2	0.0	4.1	**818.1**	**818.1**	0.0	0.1
e50.7	**865.6**	866.1	0.5	4.1	**865.6**	**865.6**	0.0	0.2
e100.1	**3507.0**	3507.9	1.1	29.7	**3507.0**	**3507.2**	0.5	16.9
e100.2	**3307.9**	3308.7	1.1	28.9	**3307.9**	**3307.9**	0.0	9.4
e100.3	**3566.3**	3566.5	0.8	25.1	**3566.3**	**3566.3**	0.0	5.0
e100.4	**3448.1**	3451.0	2.2	25.1	**3448.1**	**3448.3**	0.3	26.5
e100.5	**3637.0**	3639.4	1.9	29.5	**3637.0**	**3637.3**	0.4	31.8
e100.6	**3436.5**	3438.0	2.7	28.9	**3436.5**	**3437.3**	0.9	35.6
e100.7	**3703.5**	3704.6	2.0	28.5	**3703.5**	**3703.9**	1.1	13.8
e250.1	22089.6	22144.8	33.0	266.9	**22087.9**	**22125.6**	19.5	290.0
e250.2	22775.2	22838.6	54.6	277.2	**22770.7**	**22783.6**	17.2	269.6
e250.3	21886.1	21927.7	16.5	303	**21871.2**	**21895.5**	13.7	256.9
e250.4	23428.5	23467.6	19.9	309.5	**23422.7**	**23440.4**	12.5	259.9
e250.5	22386.9	22423.8	30.0	296.5	**22378.4**	**22395.0**	8.2	272.2
e250.6	**22285.3**	**22314.2**	22.3	377	22290.3	22314.7	9.8	298.5
e250.7	22923.9	22966.2	30.4	321	**22908.8**	**22941.7**	16.2	286.6
r100.1	**597.9**	**597.9**	0.0	16.3	**597.9**	**597.9**	0.0	0.3
r100.2	**586.0**	**586.0**	0.0	16.3	**586.0**	**586.0**	0.0	0.2
r100.3	**607.0**	**607.0**	0.0	11.1	**607.0**	**607.0**	0.0	0.2
r100.4	**598.4**	**598.4**	0.0	16.5	**598.4**	**598.4**	0.0	0.3
r100.5	**624.4**	**624.4**	0.0	16.5	**624.4**	**624.4**	0.0	0.2
r100.6	**615.5**	**615.5**	0.0	16	**615.5**	**615.5**	0.0	0.2
r100.7	**514.7**	**514.7**	0.0	14.8	**514.7**	**514.7**	0.0	0.2
r300.1	**4131.1**	**4131.1**	0.0	472.4	**4131.1**	**4131.1**	0.0	13.5
r300.2	**4040.7**	**4040.7**	0.0	307.9	**4040.7**	**4040.7**	0.0	9.7
r300.3	**4134.8**	**4134.8**	0.0	467.8	**4134.8**	**4134.8**	0.0	13.0
r300.4	**4229.3**	**4229.3**	0.0	364.7	**4229.3**	**4229.3**	0.0	11.7
r300.5	**3951.9**	**3951.9**	0.0	272.6	**3951.9**	**3951.9**	0.0	5.4
r300.6	**4314.4**	4314.5	0.0	600	**4314.4**	**4314.4**	0.0	12.8
r300.7	**4093.9**	**4093.9**	0.0	383.5	**4093.9**	**4093.9**	0.0	12.1

References

[1] D. S. Johnson, J. K. Lenstra, A. H. G. Rinnooy Kan, The complexity of the network design problem, Networks, 8 (1978) 279-285.

[2] R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Computer Networks, 52 (2008) 3229-3247.

[3] M. Fischetti, G. Lancia, P. Serafini, Exact algorithms for minimum routing cost trees, Networks, 39 (2002) 161-173.

[4] R. Wong, Worst case analysis of network design problem heuristics, SIAM Journal of Algebraic Discrete Mathematics, 1 (1980) 51-63.

[5] B. A. Julstrom, The Blob code is competitive with edgesets in genetic algorithms for the minimum routing cost spanning tree problem, Proceedings of GECCO 2005, Hans-Georg Beyer et al.. Eds., ACM Press, New York, 1 (2005) 585-590.

[6] A. Singh, A New Heuristic for the Minimum Routing Cost Spanning Tree Problem, International Conference on Information Technology (ICIT 2008), IEEE, (2008) 9-13.

[7] A. Singh, S. Sundar, An artificial bee colony algorithm for the minimum routing cost spanning tree problem, Soft Computing, 15 (2011) 2489-2499.