Tabu Search with Ejection Chain for the Biobjective Adjacent-only Quadratic Spanning Tree

Sílvia M. D. M. Maia, Elizabeth F. G. Goldbarg, Lucas D. M. dos S. Pinheiro, Marco C. Goldbarg
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Natal, Brazil
silvia@dimap.ufrn.br, beth@dimap.ufrn.br, lucasd_monteiro@outlook.com, gold@dimap.ufrn.br

***Abstract*— Given an edge-weighted simple graph *G*, the minimum quadratic spanning tree problem consists in finding a spanning tree of *G* such that the sum of the weights of its edges plus the sum of the product of the weights of pairs of edges is minimum over all spanning trees of *G*. When the product of the weights of pairs of edges is calculated only for adjacent edges, the problem is called adjacent-only minimum quadratic spanning tree. This problem belongs to NP-hard. In this study we investigate the performance of a tabu search algorithm with ejection chain for the bi-objective version of this problem. An experiment with 168 instances is reported.**

Keywords- bi-objective adjacent-only quadratic spanning tree; tabu search; metaheuristic

I. INTRODUCTION

Research on multi-objective problems has been growing in the last years once these problems are more realistic representations of complex real world applications than their single objective counterparts. Besides their wide application areas, these problems are, in general, difficult to solve by exact algorithms. Even problems for which there are polynomial algorithms to solve them when considered with one objective, become NP-hard with two or more objectives. This is the case of the Minimum Spanning Tree (MST) problem, shown to belong to NP-hard with two objectives by [1].

The problem studied here is an MST variant which belongs to NP-hard even with one objective [2]. The problem consists in finding a spanning tree *T* of an edge-weighted graph *G* such that the sum of the weights of the edges and the sum of the intercosts between adjacent edges in *T* are minimized. This problem is called Adjacent-only Minimum Quadratic Spanning Tree (AQMST) and is a particular case of the Minimum Quadratic Spanning Tree (QMST). The single objective version of this problem was presented in [2] where it was shown to belong to NP-hard. Upper and lower bounds, exact and heuristic approaches were also introduced to deal with the QMST in [2]. A genetic algorithm was proposed in [26]. It presented good results when compared to the heuristics proposed in [2]. Other genetic algorithm along with a new tree representation was provided in [23]. Exact and heuristic approaches, including tabu search and VNS, were described in [6] and [7]. Öncan and Punnen [17] devised a local search algorithm with tabu thresholding and Lagrangian relaxation for the QMST and compared the results with the algorithms in [2]. The artificial bee colony metaheuristic was applied to the QMST in [25] which was compared to the genetic algorithm of [26]. Multistart simulated annealing, hybrid genetic and iterated tabu search algorithms for the QMST were presented in [18]. Computational results indicated the iterated tabu search as the best approach. A reformulation for the AQMST that allowed stronger linear programming bounds to be computed along with computational results in favor of the proposed strategy were provided in [20] and [21]. A new model for the QMST that considers each single quadratic term together with the underlying combinatorial structure was presented in [4] and [5]. Improved bounds and processing times were claimed as results of the strategy proposed. In [9], a three-phase search algorithm, named TPS, was described for the QMST. A tabu search algorithm with strategic oscillation for the QMST was proposed in [13] and computational experiments revealed that the approach presented better results when compared with the algorithms presented in [18].

The bi-objective version of the AQMST, called bi-AQST, was first addressed in [14] where a branch-and-bound and a Pareto local search [19] were proposed. The local search algorithm was applied to instances up to 50 vertices. Evolutionary algorithms were presented in [15] such as a transgenetic [10], an NSGA-II [8] and a hybridization of former approaches denominated NSTA. Computational experiments were carried out on 132 benchmark instances. The results indicated that the transgenetic approach provided the best approximation sets.

Metaheuristics have been successful approaches to deal with multi-objective problems. Although a huge research effort has been directed to evolutionary techniques, non-evolutionary methods have also the potential to provide high quality results. This is the case of tabu search, mainly when dealing with problems with quadratic objectives such as the Quadratic Assignment Problem [16].

Ejection chain is a powerful method to create large neighborhood structures and have shown to be very effective on large scale combinatorial optimization problems [22]. Nevertheless, this approach is underexplored for multi-objective problems. In this paper, we explore the potential of the ejection chain within a tabu search algorithm for the bi-objective adjacent-only quadratic spanning tree problem. The

proposed algorithm is compared with the PLS presented in [14] and the transgenetic algorithm proposed in [15] on a computational experiment with 168 benchmark instances.

Section II presents the investigated problem. The proposed Tabu Search algorithm is presented in Section III. The computational experiment is presented in Section IV. Section V presents the conclusions and final remarks.

II. THE BI-OBJECTIVE ADJACENT-ONLY QUADRATIC SPANNING TREE

The single objective *QMST* presented in [2] is defined on a simple connected edge-weighted graph $G = (V,E)$, where $V = \{v_1,\ldots,v_n\}$ is the set of vertices and $E = \{e_1,\ldots,e_m\}$ is the set of edges. A weight w_i, $1 \le i \le m$, is assigned to each $e_i \in E$ and a cost c_{ij} is associated with each pair (e_i,e_j), $1 \le i,j \le m$, $i \ne j$, the *intercost* between edges e_i and e_j. Let T be a spanning tree of G and x_i, $1 \le i \le m$, be a binary variable such that $x_i = 1$ if $e_i \in T$. Then, the *QMST* consists in finding a spanning tree T of G such that equation 1 is minimized.

$$z(\mathbf{x}) = \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} x_i x_j \qquad (1)$$

The *AQMST* is a variant of the *QMST* for which the intercosts are evaluated only for adjacent edges, as in equation 2 where A_i denotes the set of indices of edges adjacent to e_i.

$$z(\mathbf{x}) = \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m}\sum_{j\in A_i} c_{ij} x_i x_j \qquad (2)$$

Real problems that can be modelled by the *QMST* and *AQMST* include oil derivatives transportation through pipe networks [2], transportation networks with penalties, telecommunication and energy distribution [18]. The *AQMST* was proven to be NP-hard by Assad and Xu [2] who presented a polynomial reduction of the Hamiltonian Path Problem to it.

For the *bi-AQST* the linear and the quadratic parts of equation 2 are considered independently, i.e., we seek to minimize $f_1(\mathrm{x}) = \sum_{i=1}^{m} w_i x_i$ and $f_2(\mathrm{x}) = \sum_{i=1}^{m}\sum_{j\in A_i} c_{ij} x_i x_j$.

III. THE TABU SEARCH ALGORITHM

In this section, we describe the main implementation features of a tabu search algorithm applied to the *bi-AQST*. The single objective tabu search approach was tailored to handle the multi-objective case. The adaptations were inspired by the Multi-objective Tabu Search framework (MOTS) proposed by Hansen [11]. The pseudo-code is presented in Figure 1. In an overview, the algorithm starts with the generation of a set of initial solutions, named *solSet* (line 2). The non-dominated solutions obtained during the search are stored in a global archive, named *NDSet*. This archive is initialized with the non-dominated solutions of *solSet* (line 3) and it is limited to *NDSetSize* solutions. The Adaptive Grid Archiving, introduced in [12], was used to update *NDSet*. The main loop is executed until a stopping criterion is satisfied (line 4). Computational experiments were carried out for two different stopping criteria: 1 million evaluations of the objective function (given by *maxEval*) and execution for a fixed processing time of 150s. A local search procedure, that hybridizes tabu and ejection chain strategies, explores all solutions of *solSet* (lines 5 and 6). The neighbors of each solution are evaluated and checked for inclusion in *NDSet* (line 6). In order to enhance diversification, all solutions in *solSet* are replaced with random solutions chosen from *NDSet* at intervals of a predetermined number of iterations (lines 7 and 8). The algorithm returns *NDSet* as the approximation set generated during the search (line 10). The following subsections contain the detailed explanation of the algorithm.

```
Tabu Search (solSetSize, preProcFile, maxEval,
changeInterval, NDSetSize, numOfTabuItrs)
1  itr ← 0; numofEval ← 0; NDSet ← ∅; solSet ← ∅
2  CreateInitialSolutions (solSet, solSetSize,
preProcFile)
3  InitNDSet (solSet)
4  while (stoppingCriterionIsNotSatisfied())
5    for (i ← 0; i < solSetSize; i++)
6      genAndEvalNeighbors_UpdNDSet (solSet[i], i,
itr, NDSet, numOfEval, numOfTabuItrs)
7      if (itr > 0 and itr % changeInterval = 0)
8        solSet[i] ← NDSet [random(0, NDSetSize-1)]
9    itr++
10 return NDSet;
```

Figure 1. Pseudo-code of the proposed tabu search algorithm.

A. Generation of Initial Solutions

The initial solutions are generated by the procedure described in [15] and saved in *solSet*, which is limited to *solSetSize* solutions. The method to create the initial solutions consists in a random version of the H1 heuristic, proposed originally in [2] to deal with the *QMST*. Initially, the procedure estimates l_i and q_i, $0 < i \le m$, the average contribution of each edge $e_i \in E$ to the linear and quadratic objectives, respectively. Those estimates are computed with Equations 3 and 4.

$$l_i = w_i,\ 0 < i \le m \qquad (3)$$

$$q_i = \frac{n-2}{m-1}\sum_{j\in A_i}(c_{ij} + c_{ji}),\ 0 < i \le m \qquad (4)$$

Each edge e_i is associated with a value est_i, a linear combination between l_i and q_i, i.e, the weighted estimate of the linear and quadratic average contributions of e_i. Given a weighting vector $\lambda = (\lambda_1, \lambda_2)$, where $\lambda_1, \lambda_2 \in [0, 1]$ and $\lambda_1 + \lambda_2 = 1$, est_i is calculated with equation 5 [15].

$$est_i = \lambda_1 l_i + \lambda_2 q_i \,,\ 0 < i \leq m \qquad (5)$$

The edges are sorted in non-decreasing order of est_i. After that, the creation of a new spanning tree, *st*, begins. A list of candidate edges to be in the new spanning tree in construction is created with *maxCandListSize* edges. Those edges do not induce a cycle in *st*. Each iteration, a random edge from the candidate list is chosen and inserted in *st*. The candidate list is updated by removing those edges that no longer can be included in *st* and by the inclusion of edges with the best estimates. The process is repeated until *n*-1 edges are added to *st*.

Calculating est_i and sorting the set of edges every time a new solution in *solSet* is created may be time consuming. In order to avoid such computational effort, a preprocessing phase calculates est_i and sorts the set of edges according to the following set of weighting vectors: {(0.0, 1.0), (0.1, 0.9), (0.2, 0.8), (0.3, 0.7), (0.4, 0.6), (0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2), (0.9, 0.1), (1.0, 0.0)}. The file *preProcFile* contains the ordered set of edges for each weighting vector. Consequently, the method *CreateInitialSolutions* randomly selects a sorted list of edges from *preProcFile*.

If the linear and quadratic objective values of *st* equal the objective values of some solution of *solSet*, then *st* is discarded and a new tree is randomly generated.

B. Tabu Search with Ejection Chain search strategy

Each solution of *solSet* is submitted to local search through the procedure *genAndEvalNeighbors_UpdNDSet*. The pseudo-code for this procedure is provided by Figure 2. Let *sol* be *solSet*[*i*], a solution of *solSet*. A new solution, *sol'*, is generated from *sol* by replacing edges of *sol* by edges that are not in *sol*. Several attempts of replacing one edge are performed in sequence, generating an ejection chain. Each attempt corresponds to an execution of the do-while loop in Figure 2 (lines 5-24). During this process, edges inserted in *sol'* cannot be removed and edges removed from *sol'* are no longer candidates for insertion, i.e., for a particular ejection chain, accepted replacements remain in the solution *sol'* for the next attempts.

To decide which edge must be withdrawn from *sol* to generate *sol'*, a weighting vector is randomly chosen and ec_i, the actual weighted contribution of each edge to the total cost of *sol'*, is calculated by equation 6.

$$ec_i = \lambda_1 w_i + \lambda_2 \sum_{j \in A_i \wedge e_j \in sol'} (c_{ij} + c_{ji}) \,,\ 0 < i \leq m \qquad (6)$$

ec_i is not calculated for the edges already inserted in *sol'*. The edge with greater impact to the cost of *sol'*, named *remEdge*, is chosen to be withdrawn (line 6). Once an edge is removed, the algorithm scans the edges which are candidates to be inserted in *sol'*. The algorithm calculates the variation in the weighted objective function of *sol'* due to the removal of *remEdge* and the insertion of the new edge. The variation of the value of the objective function is calculated with the same weighting vector used to choose the edge removed. The candidate edge with best variation in the objective function, *insEdge*, is chosen to be inserted in *sol'* (line 7). With *remEdge* and *insEdge* defined, *sol'* is updated (line 8).

As a result of *insEdge* choice, the new objective function value of *sol'* is already determined. In line 9, previous value for the objective function is stored and *sol'* objective function value is updated to reflect the substitution of *remEdge* by *insEdge*.

```
genAndEvalNeighbors_UpdNDSet (solSet[i], i, itr,
NDSet, numOfEval, numOfTabuItrs)
1  sol' ← solSet[i]
2  wv[0] ← rand(0.0,1.0)
3  wv[1] ← 1.0 - wv[0]
4  numFailure ← 0
5  do
6    remEdge ← chooseEdgeToRemove (wv, sol')
7    insEdge←chooseEdgeToInsert (remEdge, wv, sol')
8    changeEdge (sol', remEdge, insEdge)
9    evaluate (sol')
10   isAspiration ← addSolNDSet (sol', NDSet)
11   if (isAspiration)
12     TL[i, remEdge] ← itr + numOfTabuItrs
13     TL[i, insEdge] ← itr + numOfTabuItrs
14     numFailure ← 0
15   else
16     isTabu ← evalTabu (sol', itr, remEdge, insEdge)
17     if (not isTabu)
18       TL[i, remEdge] ← itr + numOfTabuItrs
19       TL[i, insEdge] ← itr + numOfTabuItrs
20       numFailure ← numFailure + 1
21     else
22       undoLastChanges (sol', remEdge, insEdge)
23       numFailure ← numFailure + 1
24 while (numFailure < maxFailure)
```

Figure 2. Pseudo-code of procedure to generate neighbors.

The next step of the algorithm is to decide whether or not the replacement is going to be accepted. The conditions for acceptance are: either the aspiration criterion is met or the movement is not tabu. A replacement achieves the aspiration criterion if resulting *sol'* is successfully added in *NDSet*, i.e., it is a non-dominated solution regarding *NDSet* (lines 10 and 11). In this case, tabu information is updated (lines 12 and 13) and the variable *numFailure* is set to zero (line 14). *numFailure* counts the number of sequenced iterations without generating a non-dominated solution for *NDSet*.

In case the aspiration criterion is not met, the second acceptance criterion is considered: whether or not replacing *remEdge* by *insEdge* in *sol'* is tabu (line 16). If the referred move is tabu, the last replacement performed is undone and the old objective function value is restored (lines 21 and 22). Otherwise, the replacement remains and the tabu information is updated (lines 17-19). Either way, the *numFailure* variable is incremented (lines 20 and 23).

When an edge is involved in a replacement, it is forbidden to engage in any other local search operation regarding the same solution *solSet*[*i*] for *numOfTabuItrs* iterations of tabu search algorithm (Fig. 1). Parameter *numOfTabuItrs* denotes the tabu tenure. Thus, tabu information is represented by a matrix *TL* with *solSetSize* lines and *m* columns, i.e., there is a line for every solution in *solSet* and a column for every edge in the graph. An edge *ed* can only be removed from (or inserted in) *sol'* if the current iteration of tabu search algorithm (Fig. 1), *itr*, is greater than *TL* [*i*, *ed*], where *i* represents the index of the solution *sol* in *solSet* from which *sol'* is obtained. Hence, if removing *remEdge* or inserting *insEdge* in *sol'* is tabu, the replacement move is considered tabu. This tabu strategy was adopted by Cordone and Passeri [7] to deal with the *QMST*.

The generation of the ejection chain obtained from solution *sol* in *solSet* ends when *numFailure* becomes equal to *maxFailure*. The minimum number of iterations for the main loop of the local search procedure is *maxFailure*, but the maximum number of these iterations is not known *a priori*, once the parameter *numFailure* depends on the search. The sequence of moves that generates the ejection chain is composed by interdependent movements (the replacements), once a previous replacement influences the criterion to choose which edge must be removed or inserted in the sequence. The acceptance of a move also depends on the search due to the aspiration and tabu criteria. Hence, the local search strategy proposed is dynamic and adaptive, characteristics inherent to the ejection chain approach [22].

Note that the solution *sol* (*solSet*[*i*]) in *solSet* does not change during the execution of procedure *genAndEvalNeighbors_UpdNDSet*. The tabu search approach (Fig. 1) explores the neighborhood of the same solution *solSet*[*i*] for *changeInterval* iterations. For different iterations of the algorithm, it is expected that the adoption of a tabu search approach together with the random selection of weighting vectors lead to the exploration of different areas in the search space. The mechanism used to update solutions in *solSet* is described in lines 7 and 8 of Figure 1, when these solutions are replaced by random solutions in *NDSet* after *changeInterval* iterations. The tabu algorithm proposed runs until the stopping criterion is satisfied.

IV. Computational Experiments

This section reports the results obtained by the PLS, Transgenetic and Tabu Search algorithms. The instances of the experiment are an extended set from the one available at http://homes.di.unimi.it/~cordone/research/qmst/html. The data set consists of 132 graphs, divided into 11 classes with respect to the number of vertices, $NV = \{10, 15, 20, 25, 30, 35, 40, 45, 50, 75, 100\}$. Each of those classes contains 12 graphs, generated using a combination of three characteristics: graph density (33%, 67% or 100%), range of the linear cost ([0, 10] or [0, 100]) and range of the intercost ([0, 10] or [0, 100]). The algorithms were implemented in C++, using g++ compiler 4.8.4, utilizing the -O3 flag, in a 64-bit Ubuntu 14.04 operating system. All computational experiments were performed on HP Z400 machines, with Intel Xeon QuadCore W3520 of 2.8 GHz and 8GB of RAM memory.

The proposed tabu search algorithm was compared with the local search and the transgenetic algorithm presented in [14] and [15], respectively. Each algorithm was executed independently 30 times for every instance. The experiments used either of these two stopping criteria: the maximum number of objective function evaluations, *maxEval* = 1 million, is reached or the algorithm is executed for a fixed processing time of 150 seconds. Except for small instances, the Pareto optimal sets are unknown. Reference sets were created for comparisons in those cases. The reference set related to each instance for which the Pareto optimal set is unknown was composed of nondominated solutions generated by the algorithms tested. The hypervolume quality indicator, proposed in [27], was used to assess the approximation sets generated by the algorithms. The hypervolume of the approximation set generated in each execution of a given algorithm was compared with the one from the reference set, thus obtaining a gap. This gap represents how far a given approximation set *B* is to another, say, *A*, and is calculated with equation 7, where $S(A)$ denotes the hypervolume of set *A*. The Friedman test with the post-hoc analysis of Bergmann and Hommel [3] was used to check significant differences of the experimental results. We adopted significance level 0.05. Those analyses were conducted using the MULTIPLETEST package [24].

$$Gap = \frac{S(A) - S(B)}{S(A)} \times 100 \qquad (7)$$

Tables I and II show the results of the experiments regarding quality of the approximation sets produced by the algorithms. Column *Inst* refers to the group of 12 instances with the same number of vertices. Therefore, instances with 10 vertices are analyzed as a single group, the same for instances with 15 vertices and so on. Column *#Evaluation* shows the results of the experiment in which the stopping criterion was 1 million evaluations of the objective function. Column *#Processing_Time* shows the results of the experiment in which the stopping criterion was 150s.

Table I shows two values for each algorithm. The first value is the median gap. The second value (between parentheses) is the rank of the algorithm due to the quality of its approximation set. The lower the rank, the better it is. The algorithms are ranked for each group of instances. When two algorithms exhibit the same rank, there is no statistical difference among the results produced by them.

When considering the *#Evaluation* column, the best approaches for instances up to 35 vertices, as shown in Table I, were the PLS and the tabu search, except from group 15 where the transgenetic algorithm also presented the best result. As instances grew larger, the PLS was outperformed by the tabu search and the transgenetic algorithm. As observed in the same table, the tabu search presented the overall best results. This can be explained due to the ingenious way on the choice of edges to leave and enter the solutions generated during the search phase. The superior

performance of the tabu search regarding quality of the approximation set can be observed also in Table II. This table shows the average number of solutions of the reference set found by the algorithms. The PLS behaved very similar to the tabu search on instances up to 30 nodes. As instances grew larger, the other algorithms outperformed the PLS and the tabu search presented the best performance.

TABLE I. MEDIAN GAP AND RANK

Inst	#Evaluation			#Processing_Time		
	TABU	*PLS*	*Trans*	*TABU*	*PLS*	*Trans*
10	0.00 (1)	0.05 (1)	0.25 (1)	0.00 (1)	0.05 (1)	0.25 (1)
15	0.35 (1)	2.15 (1)	0.75 (1)	0.20 (1)	2.25 (3)	1.30 (1)
20	0.10 (1)	0.95 (1)	3.50 (3)	0.10 (1)	0.80 (2)	2.35 (2)
25	0.25 (1)	0.90 (1)	5.90 (3)	0.50 (1)	0.95 (1)	5.35 (3)
30	1.30 (1)	0.80 (1)	6.05 (3)	0.35 (1)	2.30 (1)	2.30 (1)
35	0.05 (1)	4.00 (1)	8.95 (3)	0.85 (1)	0.50 (1)	8.30 (3)
40	0.00 (1)	8.75 (2)	9.45 (2)	0.90 (1)	0.50 (1)	9.30 (3)
45	0.00 (1)	18.20 (2)	7.65 (2)	0.70 (1)	0.20 (1)	6.85 (3)
50	0.00 (1)	17.75 (2)	6.50 (2)	0.65 (1)	0.65 (1)	7.70 (3)
75	0.00 (1)	45.95 (3)	7.60 (2)	0.10 (1)	2.10 (1)	7.45 (3)
100	0.00 (1)	62.40 (3)	5.30 (2)	0.00 (1)	6.65 (2)	5.00 (2)
125	0.00 (1)	79.45 (3)	5.70 (2)	0.00 (1)	24.25 (3)	5.05 (2)
150	0.00 (1)	90.90 (3)	5.55 (2)	0.00 (1)	35.85 (3)	4.70 (2)
175	0.00 (1)	93.95 (3)	4.90 (2)	0.05 (1)	31.85 (3)	4.55 (2)

The best approaches up to 75 vertices are the PLS and the tabu search algorithm, as shown on column *#Processing_Time* on table I, with the transgenetic falling behind, except on the first group. On instances from 100 vertices and beyond, the tabu search becomes the best of them, with the transgenetic coming in second, and the PLS the third. Table II shows the resemblance between the PLS and the tabu search algorithms with respect to the average number of solutions found equal to the reference set, from 10 up to 75 vertices. On the larger instances, the tabu dominates.

Table III presents the average processing times in seconds for the algorithms perform 1 million evaluations of the objective function. The quality of solutions achieved by the tabu search comes at the cost of processing times significantly higher than the PLS and the transgenetic algorithm. The huge computational effort is explained by its ability to explore the search space deeper than the other techniques. Due to the ejection chain strategy, the improvement in the results is obtained by more complex moves increasing its ability to explore farther areas of the search space.

TABLE II. AVERAGE NUMBER OF SOLUTIONS IN THE REFERENCE SET

Inst	#Evaluation			#Processing_Time		
	PLS	*TABU*	*TRANS*	*PLS*	*TABU*	*TRANS*
10	9.42	11.33	8.83	9.33	11.50	8.67
15	18.00	21.75	20.33	18.17	22.67	21.25
20	34.42	40.00	27.33	34.67	40.92	28.83
25	36.75	43.75	23.25	36.19	39.17	22.18
30	42.17	45.50	23.08	38.67	47.58	28.81
35	27.67	62.17	19.92	53.17	51.00	25.99
40	18.00	72.33	25.58	59.92	53.00	28.17
45	0.83	80.25	28.75	67.33	53.50	25.50
50	0.33	101.50	26.67	81.17	61.08	24.00
75	0.00	289.25	58.00	258.79	168.32	39.57
100	0.00	323.62	72.08	33.02	269.93	109.73
125	0.00	339.90	52.08	140.76	270.79	42.68
150	0.00	351.72	50.58	66.99	284.13	56.44
175	0.00	289.42	24.58	0.00	252.25	47.06

TABLE III. AVERAGE PROCESSING TIME IN SECONDS

Instances	Algorithms		
	PLS	*TABU*	*TRANS*
10	0.00	4.12	3.60
15	0.01	8.40	6.30
20	0.10	16.17	9.39
25	0.31	24.34	11.91
30	0.65	38.54	15.74
35	0.94	51.54	18.62
40	1.05	68.13	22.36
45	1.18	87.29	26.25
50	1.30	122.01	30.47
75	2.77	328.89	59.72
100	6.14	660.05	96.23
125	6.69	1047.89	71.50
150	13.47	1556.61	102.32
175	24.64	2279.95	141.46

V. CONCLUSIONS

We investigated the behavior of a tabu search algorithm for the bi-objective adjacent-only quadratic spanning tree problem. The local search performed by the proposed algorithm is based on ejection chain. An experiment with 168 benchmark instances was carried out. The proposed algorithm was compared to a Pareto local search and a transgenetic algorithm. For the comparison, the algorithms were given 1 million evaluations of the objective function on the first set of experiments, and 150 seconds of time on the second. The results of the experiments indicated that, as the

instances grow larger, the tabu search becomes the most competitive approach for the problem.

ACKNOWLEDGMENT

This work was partially supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil, under Grants 301845/2013-1 and 302819/2011-8.

REFERENCES

[1] V. Aggarwal, Y. Aneja, and K. Nair, "Minimal spanning tree subject to a side constraint," Computers & Operations Research, vol. 9, pp. 287-296, 1982.

[2] A. Assad, and W. Xu, "The quadratic minimum spanning tree problem," Naval Research Logistics, vol. 39, pp. 399-417, 1992.

[3] G. Bergmann, G. Hommel, Improvements of general multiple test procedures for redundant systems of hypotheses, in: P. Bauer, G. Hommel, E. Sonnemann (Eds.), Multiple Hypotheses Testing, Springer, 1988, pp. 100–115.

[4] C. Buchheim, and L. Klein, "The spanning tree problem with one quadratic term," Proceedings of 12th Cologne - Twente Workshop on Graphs and Combinatorial Optimization, pp.31-34, 2013.

[5] C. Buchheim, and L. Klein, "Combinatorial optimization with one quadratic term: spanning trees and forests," Discrete Applied Mathematics, vol. 177, pp.34-52, 2014.

[6] R. Cordone, and G. Passeri, "Heuristic and exact approaches to the quadratic minimum spanning tree problem," Proceedings of Seventh Cologne - Twente Workshop on Graphs and Combinatorial Optimization, pp.52-55, 2008.

[7] R. Cordone, and G. Passeri, "Solving the quadratic minimum spanning tree problem," Applied Mathematics and Computation, vol. 218, pp. 11597-11612, 2012.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, n. 2, pp.182-197, 2002.

[9] Z. Fu, and J. Hao, "A three-phase search approach for the quadratic minimum spanning tree problem," Engineering Applications of Artificial Intelligence, vol. 46, pp. 113-130, 2015.

[10] E. F. G. Goldbarg, and M. C. Goldbarg, "Transgenetic algorithms: a new endosymbiotic approach for evolutionary algorithms", in: A. Abraham, A-E. Hassanien, P. Siarry, and A. Engelbrecht, (eds.) Foundations of Computational Intelligence, vol. 3, pp.425-460, 2009.

[11] M. P. Hansen, "Tabu search for multiobjective optimization: Mots," Proceedings of the Thirteenth International Conference on Multiple Criteria Decision Making, pp. 6-10, 1997.

[12] J. D. Knowles, "Local-search and hybrid evolutionary algorithms for Pareto optimization," PhD thesis, Department of Computer Science, University of Reading, Reading, UK, 2002.

[13] M. Lozano, F. Glover, C. García-Martinez, F.J. Rodríguez, and R. Martí, "Tabu search with strategic oscillation for the quadratic minimum spanning tree," IIE Transactions, vol. 46, n. 4, pp.414-428, 2014.

[14] S. M. D. M. Maia, E. F. G. Goldbarg, M. C. Goldbarg, "On the biobjective adjacent only quadratic spanning tree problem," Electronic Notes in Discrete Mathematics, vol. 41, pp. 535-542, 2013.

[15] S. M. D. M. Maia, E. F. G. Goldbarg, M. C. Goldbarg, "Evolutionary algorithms for the bi-objective adjacent only quadratic spanning tree," International Journal of Innovative Computing and Applications, vol. 6, n. 2, pp. 63-72, 2014.

[16] A. Misevicius, "An implementation of the iterated tabu search algorithm for the quadratic assignment problem," OR Spectrum, vol. 32, n. 3, pp. 665-690, 2012.

[17] T. Öncan, and A. P. Punnen, "The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm," Computers and Operations Research, vol. 37, n. 10, pp.1762-1773, 2010.

[18] G. Palubeckis, D. Rubliauskas, and Targamadzé, A. "Metaheuristic approaches for the quadratic minimum spanning tree problem," Information Technology and Control, vol. 39, n. 4, pp.257-268, 2010.

[19] L. Paquete, and T. Stützle, "Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: A Review," Technical report TR/IRIDIA/2006-001, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle Université Libre de Bruxelles, 2006.

[20] D. L. Pereira, M. Gendreau, and A. S. Cunha, "Stronger lower bounds for the quadratic minimum spanning tree problem with adjacency costs," Electronic Notes in Discrete Mathematics, vol. 41, n. 5, pp.229-236, 2013.

[21] D. L. Pereira, M. Gendreau, and A. S. Cunha, "Lower bounds and exact algorithms for the quadratic minimum spanning tree problem," Computers & Operations Research, vol. 63, pp.149-160, 2015.

[22] C. Rego, T. James, and F. Glover, "An ejection chain algorithm for the quadratic assignment problem," Networks, vol. 56, n. 3, pp. 188-206, 2010.

[23] S. M. Soak, D. W. Corne, and B. H. Ahn, "The edge-window-decoder representation for tree based problems," IEEE Transactions on Evolutionary Computation, vol. 10, n. 2, pp.124-144, 2006.

[24] Statistical Inference in Computational Intelligence and Data Mining, http://sci2s.ugr.es/sicidm/.

[25] S. Sundar, and A. Singh, "A swarm intelligence approach to the quadratic minimum spanning tree problem," Information Sciences, vol. 180, n. 17, pp.3182-3191, 2010.

[26] G. Zhou, and M. Gen, "An effective genetic algorithm to the quadratic minimum spanning tree problem," Computers & OR, vol. 25, n. 3, pp.229-237, 1998.

[27] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: methods and applications," PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 1999.