Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-3 (2015) 1167–1172

Tabu Search Heuristics for Two Bounded Degree Spanning Tree Problems ⋆

Temel Öncan *

* *Endüstri Mühendisliği Bölümü, Galatasaray Üniversitesi, Ortaköy, İstanbul, 34357, TÜRKİYE (e-mail: ytoncan@gsu.edu.tr).*

Abstract: In this work, we address two variants of the spanning tree problem. The first problem deals with finding a spanning tree such that the number of branch vertices, namely vertices with degrees of at least three, is minimum (MBV) and the second one involves constructing a spanning tree with minimum sum of branch vertex degrees (MDS). We suggest an attribute based Tabu Search (TS) heuristic approach for both problems. According to extensive computational experiments done on standard test sets, we observe that the TS heuristics outperform the best known heuristics from the literature.

Keywords: tree structure; spanning tree; branch vertex; heuristic; tabu search

1. INTRODUCTION

Many combinatorial optimization problems consist of constructing *spanning trees* which are optimal with respect to some conditions. For a discussion of various real-life applications of the spanning trees arising in transportation, telecommunication, energy distribution, irrigation, data storage and cluster analysis we refer to the book by Ahuja et al. (1993).

In this paper we address two variants of the spanning tree problem: the spanning tree problem with a minimum number of branch vertices, i.e. vertices with degrees of at least three, (MBV) and the spanning tree problem with a minimum sum of branch vertex degrees (MDS), which are defined as follows. Let $G = (V, E)$ be a connected undirected graph with a set of vertices $V = \{1, \ldots, n\}$ and a set of edges $E = \{e_1, \ldots, e_m\}$ where n and m stand for the number of vertices and edges, respectively. A spanning tree of an undirected graph G is a subgraph that includes all of the vertices of G which forms a tree. Then, the MBV involves finding a spanning tree of G such that the number of branch vertices is minimum and the MDS consists of constructing a spanning tree of G with minimum sum of branch vertex degrees. Notice that MBV and MDS are related problems. The MBV was first introduced by Gargano et al. (2002) in their seminal work whereby they showed it to be NP-complete. Later on, Cerulli et al. (2009) have introduced the MDS and proved its NP-completeness. The authors also suggested Single Commodity Flow (SCF) formulations for each, then they devised three construction heuristics. Recently, Sundar et al. (2012) suggested efficient construction heuristics and hybrid Ant Colony Optimization (ACO) algorithms and, Carrabs et al. (2013) devised lower and upper bounding algorithms for both MBV and MDS.

Real-life applications of MBV and MDS arise prominently in Wavelength Division Multiplexing (WDM) technology, used in optical networks (Stern and Bala 1999. The WDM technology enables the fiber optic communication used for web browsing, video conferences, video on demand services, etc. (Gargano et al. 2002). Fiber optic communication encodes information into light waves and beams it through an optical fiber. Since the wavelength of light determines its characteristic, each wavelength carries its own independent data-traffic. In WDM systems, several different wavelengths are combined and simultaneously transmitted over a single fiber. A multiplexer is used to join signals at the transmitter (i.e. source vertex) whereas a de-multiplexer, at the receiver (i.e. destination vertex) splits them apart. Furthermore, WDM technology also permits multi-casting in an optical network by using sophisticated switches which make copies of optical signals by splitting light, and hence, they transmit information from a single source vertex to multiple destination vertices. The multi-casting technique enables the distribution of copied data packets to multiple users via sophisticated light-splitting switches, located on the branch vertices of the optical network. Therefore, it is of great interest to design optical networks which enable multi-casting with a minimum number of light-splitting switches and with a minimum sum of branch vertex degrees, taking into account their costs (Sundar et al. 2012).

Considering the fact that both MBV and MDS are NP-complete, it is difficult to get their optimal solutions. The motivation of this study is to suggest an accurate and efficient Tabu Search (TS) algorithms for both problems.

The remainder of this work is organized as follows. In the next section, the TS heuristics are introduced. In Section 3 we report the results of our extensive computational experiments. Finally, we conclude the paper with Section 4.

⋆ This research is supported by the Turkish Scientific and Technological Research Council Research Grants Nos. 107M462 and 109M139, and Galatasaray University Scientific Research Project Grant No 14.402.003.

10.1016/j.ifacol.2015.06.242

2. TABU SEARCH

Starting from an initial one, the TS algorithm moves at each iteration from the current solution to the best one in a subset of its neighborhood, even if this causes a deterioration in the objective function value (Glover and Laguna, 1997). In order to avoid cycling, solutions possessing some attributes of recently visited solutions are declared *tabu* for a certain number of iterations, named the tabu tenure. The algorithm stops whenever a preset criterion is satisfied. When an attribute is declared tabu, all solutions possessing this attribute are implicitly declared *tabu* as well. However, some of these solutions may never be considered by the search. To remedy this, an aspiration criterion is defined to override the tabu status of a solution. One common aspiration criterion is to allow tabu solutions yielding better solutions than that of the best known solution.

In our TS, we associate an attribute to each edge $e_p \in E$ of the graph G. Hence, the attribute set $\Lambda(s)$ of solution s, is defined as the set of edges selected from the current spanning tree. We define the tabu rule as follows. If edge e_p is removed from the spanning tree, then reinserting it into the spanning tree is forbidden for the next ω iterations, where ω is the tabu tenure. The last iteration for which attribute e_p is declared tabu is denoted by μ_p. The tabu status of an attribute can be revoked if it leads to a solution with smaller cost than that of the best solution identified having that attribute. The aspiration level ϑ_p of attribute e_p is initially set to the cost of the initial solution s_0 if edge e_p belongs to this solution, and to ∞ otherwise. At every iteration, the aspiration level of each attribute $e_p \in \Lambda(s)$ of the current solution is updated to $\min\{\vartheta_p, f(s)\}$, where $f(s)$ stands for the cost value of solution s. Given a solution s, its neighbor solutions are defined by *N(s)*. At each iteration, we consider a subset $T(s) \subseteq N(s)$ which consists of all solutions $\overline{s} \in N(s)$ reachable from s without incurring the risk of cycling, i.e., those that are not *tabu* or that satisfy the aspiration criterion. Moreover, whenever the current solution s is better than the best known solution s^*, i.e. $f(s) < f(s^*)$ holds, then the best known solution s^* is updated.

During the run of the TS algorithm, we maintain a *frequency array* ν, which records the number of times a particular edge appears in the solutions visited by the TS algorithm. For example, if $\nu_p = 5$, then in all solutions considered so far, edge e_p is included 5 times.

In order to perform additional intensification with path relinking, we also maintain a *recency array* ζ. The recency array ζ is updated whenever TS obtains a local optimal solution s'. That is to say ζ_p is increased by 1 if the local optimal solution s' includes edge e_p. In other words, the recency array ζ tracks the most recent ψ best local minimums where ψ is a predefined parameter.

2.1 Neighborhoods for MBV and MDS

Let Δ_i denote the set of edges incident to vertex i in G, and $|\Delta_i|$ stands for the degree of the vertex i. Furthermore, let Γ_i be the set of edges incident to vertex i in the current spanning tree s, and $|\Gamma_i|$ indicates the cardinality of the set

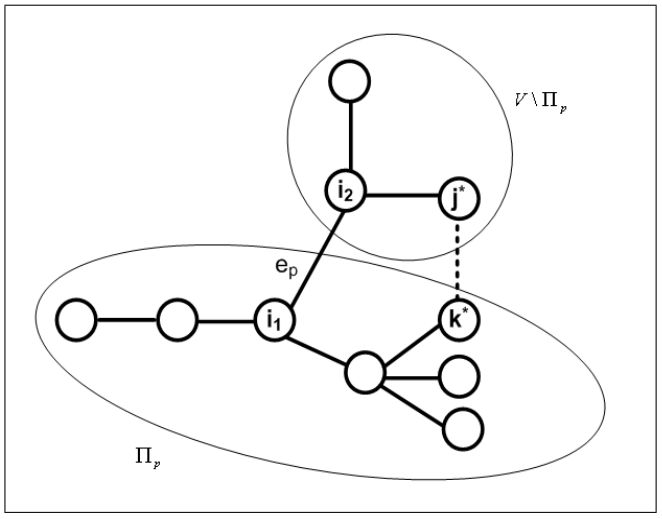

Fig. 1. An illustration of the neighborhood $N^1(s)$

Γ_i, namely the local degree of vertex i. Note that, $\Gamma_i \subseteq \Delta_i$ and hence $|\Gamma_i| \leq |\Delta_i|$ holds for all branch vertices.

For both MBV and MDS, we propose the c-edges exchange neighborhood $N^c(s)$ as follows. First, we select a branch vertex i such that $|\Delta_i| > c$ holds. Considering all edges incident to the branch vertex i we remove c of them, and then, we simultaneously try to insert c edges different from the ones removed without damaging the spanning tree structure. Taking into consideration all possible combinations of choosing c out of $|\Delta_i|$ edges incident to branch vertex i, we replace c edges incident to branch vertex i with the new ones considering the following *insertion conditions*.

(1) The inserted edges yield an improvement in the objective function value.
(2) In case the first rule fails we insert non-tabu edges with minimum frequencies (i.e. the edges selected with the least number of times in the solutions visited by the TS algorithm). Ties are broken arbitrarily.

For instance, the neighborhood $N^1(s)$ is defined as follows. First, we choose a branch vertex i_1. Then, we consider replacing edge $e_p \in E \cap \Delta_{i_1} \cap \Lambda(s)$, which connects vertices i_1 and i_2, with another edge without harming the spanning tree structure. Note that, when edge e_p is removed we obtain two disconnected spanning trees or components. Let Π_p stand for the set of vertices corresponding to the component disconnected from the rest of vertices such that $i_1 \in \Pi_p$ and $i_2 \in V \setminus \Pi_p$ hold. Then we find an edge which is not in the current solution and which connects two components Π_p and $V \setminus \Pi_p$, in order to re-construct a spanning tree, taking into account the insertion conditions stated above.

Finally, among all $|\Delta_{i_1}|$ edges incident to branch vertex i_1, we replace edge $e_p^* \in E \cap \Delta_i \cap \Lambda(s)$, with another edge say $\{j^*, k^*\} \in E$ such that $j^* \in V \setminus \Pi_p$ and $k^* \in \Pi_p$, considering the above mentioned insertion conditions.

Figure 1 illustrates an example of the neighborhood $N^1(s)$ for MBV and MDS. When an edge e_p incident to branch vertex i_1 with $|\Gamma_{i_1}| = 3$ is removed, we obtain two disconnected components and hence two vertex sets Π_p

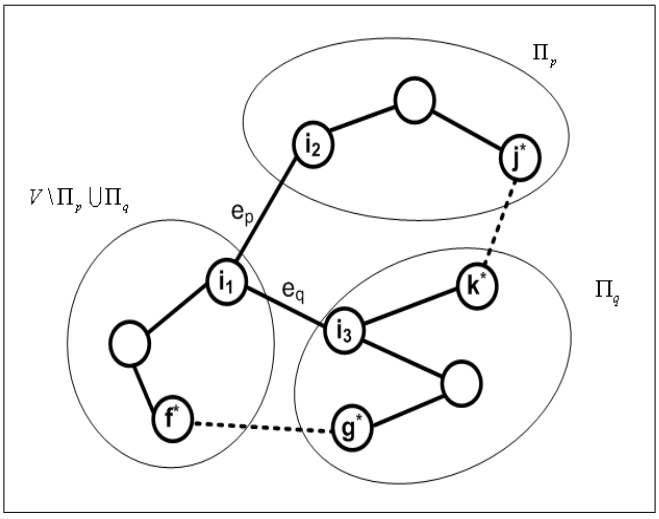

Fig. 2. An illustration of the neighborhood $N^2(s)$

and $V \setminus \Pi_p$ such that $i_1 \in \Pi_p$ and $i_2 \in V \setminus \Pi_p$. Then among all candidate edges $\{j,k\} \in E$ with $j \in V \setminus \Pi_p$ and $k \in \Pi_p$, we insert edge $\{j^*,k^*\}$.

Now we describe the neighborhood $N^2(s)$. First we select a branch vertex i_1 and strive to replace two edges e_p and e_q, corresponding to $\{i_1,i_2\}$ and $\{i_1,i_3\}$ respectively, with two other edges, without harming the spanning tree structure. Note that whenever we remove edges e_p and e_q, we obtain three disconnected components. Let Π_p and Π_q stand for the vertex sets which correspond to the components disconnected from the rest of vertices by removing edges e_p and e_q such that $i_2 \in \Pi_p$ and $i_3 \in \Pi_q$, respectively. Then among all candidate edges which are not in the current solution, i.e. $e \in E \setminus \Lambda(s)$, we consider two of them which connect components Π_p and $V \setminus \Pi_p \cup \Pi_q$; Π_q and $V \setminus \Pi_p \cup \Pi_q$; and Π_p and Π_q. Assume the candidate edges are $\{h,l\}$, $\{f,g\}$, $\{j,k\} \in E \setminus \Lambda(s)$ such that $j,l \in \Pi_p$, $g,k \in \Pi_q$, and $f,h \in V \setminus \Pi_p \cup \Pi_q$. Thus, we have no choice but to insert one of the following edge pairs $\{h,l\}$, $\{f,g\}$; $\{h,l\}$, $\{j,k\}$; $\{f,g\}$, $\{j,k\}$ in order to re-establish a spanning tree. Note that vertices l, g and f may not be necessarily distinct from vertices j, k and h, respectively. Consequently, among all candidate edge pairs we insert the one considering the insertion conditions.

In order to better expose the neighborhood $N^2(s)$, Figure 2 depicts an illustrative example for MBV and MDS. Here, edges e_p and e_q, which are incident to branch vertex i_1 with $|\Gamma_{i_1}| = 3$, are replaced with edges $\{j^*,k^*\}$ and $\{f^*,g^*\}$, respectively. Observe that removing edges e_p and e_q yields three disconnected components and hence three distinct vertex sets Π_p, Π_q and $V \setminus \Pi_p \cup \Pi_q$ such that $i_1 \in V \setminus \Pi_p \cup \Pi_q$, $i_2 \in \Pi_p$ and $i_3 \in \Pi_q$. Then among all candidate edges, namely $\{h,l\}$, $\{f,g\}$, $\{j,k\} \in E \setminus \Lambda(s)$ such that $j,l \in \Pi_p$, $g,k \in \Pi_q$, and $f,h \in V \setminus \Pi_p \cup \Pi_q$, edges $\{j^*,k^*\}$ and $\{f^*,g^*\}$ are inserted.

For the MBV the neighborhood $N^1(s)$, outputs an improved solution for a branch vertex i_1 with $|\Gamma_{i_1}| = 3$ when the inserted edge $\{j^*,k^*\}$ satisfies $|\Gamma_{j^*}| \neq 2$ and $|\Gamma_{k^*}| \neq 2$. Moreover, given a branch vertex i_1 with $|\Gamma_{i_1}| = 3$ or $|\Gamma_{i_1}| = 4$, the neighborhood $N^2(s)$ yields an improved solution when the inserted edges $\{j^*,k^*\}$ and $\{f^*,g^*\}$ satisfy $|\Gamma_{j^*}| \neq 2$, $|\Gamma_{k^*}| \neq 2$, $|\Gamma_{f^*}| \neq 2$ and $|\Gamma_{g^*}| \neq 2$.

On the other hand, for the MDS, the neighborhood $N^1(s)$ yields an improved solution for a branch vertex i_1 with $|\Gamma_{i_1}| > 2$ when we insert an edge $\{j^*,k^*\}$ with $|\Gamma_{j^*}| + |\Gamma_{k^*}| < |\Gamma_{i_1}|$. Furthermore, we obtain an improved solution using the neighborhood $N^2(s)$, for a branch vertex i_1 with $|\Gamma_{i_1}| > 2$ when one of the following conditions of the neighborhood $N^2(s)$, is satisfied for the inserted edges $\{j^*,k^*\}$ and $\{f^*,g^*\}$:

- Only one of the local degrees $|\Gamma_{j^*}|$, $|\Gamma_{k^*}|$, $|\Gamma_{f^*}|$ and $|\Gamma_{g^*}|$ is larger than 2 and three others are equal to 1
- Only one of the local degrees $|\Gamma_{j^*}|$, $|\Gamma_{k^*}|$, $|\Gamma_{f^*}|$ and $|\Gamma_{g^*}|$ is equal to 1 and three others are larger than 2
- $|\Gamma_{j^*}| = |\Gamma_{k^*}| = 1, |\Gamma_{f^*}| > 2$ and $|\Gamma_{g^*}| > 2$
- $|\Gamma_{j^*}| > 2; |\Gamma_{k^*}| > 2, |\Gamma_{f^*}| = 1$ and $|\Gamma_{g^*}| = 1$
- $|\Gamma_{j^*}| = |\Gamma_{k^*}| = |\Gamma_{f^*}| = |\Gamma_{g^*}| = 1$
- All of the local degrees $|\Gamma_{j^*}|$, $|\Gamma_{k^*}|$, $|\Gamma_{f^*}|$ and $|\Gamma_{g^*}|$ are larger than 2.

Clearly larger values of c results in a wider search space. In our TS algorithm we employ the $N^1(s)$ and $N^2(s)$ neighborhoods consecutively following the sequence $N^1(s)$, $N^2(s)$, $N^1(s)$, $N^2(s)$,... We first start with neighborhood $N^1(s)$ and apply it until there is no improvement in the solution for t_1 iterations. Then we switch to neighborhood $N^2(s)$ and use it until no further improvement is obtained for t_2 iterations. Next, we switch back to neighborhood $N^1(s)$. This process is repeated until the algorithm satisfies the stopping criterion. We have chosen $t_1 = 100$ and $t_2 = 40$, which seemed to work best for most of the instances.

2.2 Attribute based tabu search algorithm

We now describe the steps of the TS algorithm which are the same for both MBV and MDS. However, their neighborhood structures and parameter settings are different. In order to construct an initial feasible solution we employ the heuristics designed by Sundar et al. (2012) called Heu_MBV for MBV and Heu_MDS for MDS. The notation used is provided in Table 1.

Table 1. Notation used in the TS algorithm

$s, \bar{s}$	: solutions, i.e. spanning trees
s'	: best solution identified in $T(s)$
s^*	: best feasible solution identified during the run of TS
$s^\#$	: the best solution found at the end of κ consecutive iterations without improving s^*
$f(s)$	: cost value for solution s
$p(s)$	: total penalized cost for solution s
$\Lambda(s)$	: attribute set for solution s
$N(s)$	: neighborhood for solution s
$T(s)$	: a subset of neighborhood for solution s
ω	: tabu tenure
τ	: iteration counter
φ	: iterations without improving the solution value
η_e	: tabu status of the removed edge e_p
κ	: number of consecutive iterations without improving s^*
ϕ	: penalty factor used to adjust the intensity of the diversification
ψ	: number of recent best local solutions
ϑ_p	: aspiration level of attribute e_p
ζ	: recency array
ν	: frequency array
ν_p	: number of times attribute e_p has been added to the solution
ρ_i	: number of times branch vertex i has been considered
μ_p	: last iteration for which attribute e_p is declared tabu

We first initialize the number of times attribute e_p has been added into the solution. This parameter is set to 1 for all edges in the current solution (i.e., $\nu_p = 1$ for all $e_p \in \Lambda(s)$, and $\nu_p = 0$ for all $e_p \notin \Lambda(s)$). Then, the last iteration for which attribute e_p is declared as tabu is set to 0 for all edges (i.e., $\mu_p = 0$ for all $e_p \in E$). The aspiration levels ϑ_p are set to the initial solution value $f(s)$ for all edges in the current solution (i.e., $\vartheta_p = f(s)$ for all $e_p \in \Lambda(s)$ and $\vartheta_p = \infty$ for all $e_p \notin \Lambda(s)$). The number of times branch vertex i has been considered for the neighborhood search is set to 0 for all vertices with degree at least three (i.e., $\rho_i = 0$ for all $i \in V$ and $|\Delta_i| > 2$). Finally, the iteration counter τ is set to 0.

The first step of our neighborhood scheme is the branch vertex selection procedure which is crucial for the performance of the neighborhoods $N^1(s)$ and $N^2(s)$, and hence to the performance of the TS algorithm. As a simple rule, we can randomly select branch vertices. However, during the run of the TS algorithm this may cause the algorithm to repeatedly visit the same branch vertices and hence the same search space without improving the solution. Especially, when the TS algorithm runs for a small number of iterations, some branch vertices are selected less often than others. Therefore, we propose a diversification scheme for the branch vertex selection considering the branch vertex visit frequencies, namely ρ_i, which denotes the number of times a branch vertex i has been considered. The branch vertex selection is performed with probabilities proportional to the inverse of the ρ_i values. Thus, the lower the value of ρ_i, the higher the probability of selecting branch vertex i. When all vertices $i \in V$ with $|\Delta_i| > 2$ have been visited an equal number of times, we apply the random selection rule for at most φ iterations without improving the solution value. If the random selection of branch vertices has not yielded an improved solution for φ iterations, we switch back to branch vertex selection by visit frequencies rule, and so on.

The next step after branch selection is to evaluate a subset $T(s)$ of neighbor solutions. For this purpose, we first initialize $T(s)$ with an empty set. Then, for each neighbor solution $\overline{s}$ of the current solution s, if there exists an attribute $e_p \in \Lambda(\overline{s}) \backslash \Lambda(s)$ such that $\mu_p < \tau$ or $f(\overline{s}) < \vartheta_p$, we set $T(s) = T(s) \cup \{\overline{s}\}$. In other words, we add to $T(s)$ the neighbor solutions $\overline{s}$ such that either $\overline{s}$ is non-tabu or the cost of $\overline{s}$ is lower than the aspiration level ϑ_p of one of the attributes $e_p \in \Lambda(\overline{s})$. To penalize the edges frequently added into the solution, we define a penalized cost value for each solution $\overline{s}$ in the subset of neighborhoods $T(s)$ as follows:

$$p(\overline{s}) = \begin{cases} f(\overline{s}) + \phi f(s)(3\sqrt{n} + \sqrt{m}) \sum_{e_p \in A(\overline{s}) \backslash \Lambda(s)} \nu_p/\tau, \\ \qquad \text{if} \quad c(\overline{s}) \geq f(s) \\ f(\overline{s}), \qquad \text{otherwise.} \end{cases}$$

Note that a penalty, $\phi f(s)(3\sqrt{n} + \sqrt{m}) \sum_{e_p \in A(\overline{s}) \backslash \Lambda(s)} \nu_p/\tau$ is added to $f(\overline{s})$ for a neighbor solution which does not yield an improvement to the objective function value. Here, ϕ is a factor used to adjust the intensity of the diversification, while the square root factor $3\sqrt{n} + \sqrt{m}$, is used to compensate for instance size. The structure of the penalized cost function $p(\overline{s})$ is similar to the one used in Taillard (1993) in the context of the VRP. Finally, considering all solutions $\overline{s}$ in $T(s)$ we identify a neighbor solution s' with the minimum penalized cost value $p(s')$.

Considering the new solution s' we update the following parameters. First, we update the tabu status of the removed edges $e_p \in \Lambda(s) \setminus \Lambda(s')$ in order to forbid them from entering the solution for at least ω iterations (i.e., $\mu_p = \tau + \omega$). Second, for each attribute $i \in \Lambda(s') \setminus \Lambda(s)$, the number of times attribute e_p has been added to the solution is increased by one (i.e., $\nu_p = \nu_p + 1$). Third, if the new solution is better than the current best solution (i.e., $f(s') < f(s^*)$) we update the current best solution (i.e., $s^* = s'$). Furthermore, in case the new solution s' is a local optimal solution then ζ_p is increased by 1 if edge e_p is in s'. Finally, for each edge of the new solution we update its aspiration level (i.e., $\vartheta_p = \min\{\vartheta_p, f(s')\}$ for all $e_p \in \Lambda(s')$).

During the run of the TS algorithm, we apply an intensification strategy with path relinking after κ consecutive iterations without improving the best known solution s^*. The path relinking approach was first developed by Glover and Laguna (1997) in the context of TS. We apply this procedure to the best solution found $s^{\#}$ at the end of κ consecutive iterations without improvement, which is not necessarily equal to the best solution s^*. The path relinking procedure tries to perform moves in the direction of a set of *elite* solutions found, namely ψ best local minimums, during the run of the TS algorithm. After having κ consecutive TS iterations without improvement, the path relinking procedure builds an auxiliary subgraph G_s of G by considering the recency array ζ. The auxiliary subgraph G_s of G is constructed such that G_s has all vertices of G, but only the edges corresponding to the non-zero entries of ζ. Furthermore, we sort the entries of the frequency array ν and add the first third edges to G_s, plus the edges in the best solution s^* so far. Then, considering the auxiliary subgraph G_s, we apply a greedy local search procedure. The greedy local search procedure is performed by considering branch vertices. This procedure first fixes a visit sequence of branch vertices. The branch vertices are sequenced in increasing order of their local degrees. Then the greedy local search procedure applies both neighborhoods $N^1(s)$ and $N^2(s)$ to each branch vertex according to the fixed sequence. We proceed in that way as long as we obtain improved solutions. If the solution returned is better than the best solution s^*, it is updated. After performing the path relinking process we continue to run the TS.

Several stopping criteria can be adopted for the TS algorithm, such as a time limit, an iteration limit or an absence of improvement for a given number of iterations. We choose our criterion so as to achieve the best compromise between the computation time and solution quality. The TS algorithm stops when the iteration counter τ reaches the iteration limit η. Furthermore, the TS is stopped when $f(s^*) = 0$ is satisfied.

3. COMPUTATIONAL RESULTS

In this section, we present the details of our computational experiments with the TS algorithm which was implemented in C++ and run on Dell Server PE2900 with two 3.16 GHz Quad Core Processors and 32 GB RAM.

First of all, we describe the test bed, then introduce parameter calibration of the TS algorithm and finally discuss a summary of the computational experiments.

3.1 Test Bed

To perform our experiments, we employed the test instances generated by Sundar et al. (2012). These instances were generated using the instance generators *Netgen*, *Genmax* and *Random* as suggested by Cerulli et al. (2009). Hence, our test bed consists of three test instance classes named Netgen, Genmax and Random. For example, Netgen stands for the class of instances output by the *Netgen* generator. Sundar et al. (2012) consider different values for number of vertices, namely they have set $n = 20, 30, 40, 50, 100, 300, 500, 1000$. Moreover, for each value of n, they generated instances with various densities: $d = 1.5, 2, 4, 10, 15$. Density of an instance is defined as the ratio of number of edges m to number of vertices n, i.e. $d = \frac{m}{n}$. Note that, the number of vertices n and densities d for each class of instances constitute $8 \times 5 = 40$ combinations. However, Sundar et al. (2012) state that the *Random* generator was not able to generate instances with $n = 20, d = 10$; $n = 20, d = 15$ and $n = 30, d = 15$ due to some internal restrictions. Therefore, for all test classes except the Random class, they generated 40 combinations of n and d. However, 37 combinations of n and d are produced for the Random class. Furthermore, they generated 5 instances for each combination of n and d, which make $40 \times 5 = 200$ instances for both the Netgen and Genmax classes, and $37 \times 5 = 185$ instances for the Random class. Consequently, we have 385 instances in total for MBV and MDS. All instances are available upon request.

3.2 Parameter calibration

In order to obtain promising results, the fine tuning of parameters used within the TS heuristic must be carefully handled. For both MBV and MDS, we calibrated the parameters considering Genmax instances with vertex sizes $50 \leq n \leq 300$.

First of all, we focused on calibrating the TS parameters for the MBV. For that purpose, we first fixed $\omega = 9$, $\varphi = 50$ and $\eta = 5000$ and performed tests on all instances using different values of parameter ϕ, which is used to adjust the intensity of the diversification, in the interval $[0.01, 0.05]$ with increments of 0.01. We observe in most instances the best value of parameter ϕ lies in the interval $[0.01, 0.03]$. Hence we determine $\phi = 0.02$ is the most appropriate value. To settle on the most appropriate value for ω we performed experiments by fixing other parameters (namely, $\phi = 0.02$, $\varphi = 50$ and $\eta = 5000$). Generally, setting parameter $\omega = 11$ seems appropriate for most instances. However, the proper value for this parameter also seems to depend on instance size. We thus performed experiments by choosing the value of ω from the interval $[\lceil \log_{10}(n+m) \rceil, \lceil 10 \log_{10}(n+m) \rceil]$. For most instances we observed when the tabu duration ω is fixed to $\lceil 3 \log_{10}(n+m) \rceil$ we get better results. In order to control diversification of the branch vertex selection process, we use the φ parameter. In this phase we successively apply two rules: random branch vertex selection and branch vertex selection according to the frequency. At the beginning of the algorithm the branch vertices are randomly selected. Then, after φ iterations without improving the solution value, we switch to the branch vertex selection by visit frequency. In our preliminary experiments, we observed that the most appropriate value for φ is 100. For values larger than 100 we observed that the algorithm wastes too much time by selecting recently visited branch vertices. On the other hand, when we set lower values, we observe that branch vertices are visited sequentially, since the visit frequencies of branch vertices become almost equal to each other. We set $\eta = 40(n + \frac{60n}{m})$ to reach a balance between solution quality and CPU time. To adjust the intensification procedure with path relinking, we set κ and ψ to $\lceil \sqrt{n} + 20\frac{m}{n} \rceil$ and $\lceil \frac{m}{2n} \rceil + 2$, respectively.

The TS algorithm parameters for the MDS are fine-tuned just as the ones for the MBV. We performed preliminary experiments to calibrate our parameters on the Genmax instances with vertex sizes $50 \leq n \leq 300$. For the MDS, we set $\phi = 0.1$, $\varphi = 40$, $\eta = 50(n + \frac{90n}{m})$ as the most appropriate values. The tabu tenure parameter ω is chosen as $\lceil 4 \log_{10}(n+m) \rceil$. Finally, κ and ψ are set to $\lceil \sqrt{n} + 30\frac{m}{n} \rceil$ and $\lceil \frac{m}{1.5n} \rceil + 2$, respectively.

3.3 Summary of the computational experiments

The computational results on the performance of the proposed heuristic and lower bounding procedures for MBV and MDS, are provided in Table 2, Table 3 and Table 4 for the Genmax, Netgen and Random classes, respectively. In all tables, the last rows include column averages and the first two columns include combinations of n and d. Column headings "UB" indicate the upper bound values obtained. The computational times in seconds are given under the columns "CPU". In all tables the first part consists of 3 columns, devoted to the MBV. The second part includes 3 columns for MDS. The columns labelled "BEST" give the best known solutions in the literature. The best known solutions are taken from results obtained by hybrid ACO algorithms by Sundar et al. (2012). Under the "TS" columns we report upper bounds and CPU times in seconds required for the TS algorithm proposed. As observed in Table 2, Table 3 and Table 4, in all cases the TS algorithm yields the best known solutions. Note that, the TS heuristics improved the best known solutions for 55 out of 385 MBV instances and 31 out of 385 MDS instances. Although, the attribute based TS heuristics remain promising in terms of accuracy, they are clearly more costly than the hybrid ACO algorithms proposed by Sundar et al. (2012).

For the sake of completeness, we should also mention the CPU times reported by Sundar et al. (2012) for their hybrid ACO algorithms. Since the results are obtained on different platforms, the computational times cannot be directly compared. Results for the hybrid ACO algorithms by Sundar et al. (2012) were obtained on a Linux based 3.2 GHz Pentium 4 computer with 1 GB RAM. The hybrid ACO algorithm for the MBV required on average 67.43, 20.28 and 76.95 seconds for the Genmax, Netgen and Random classes, respectively. On the other hand, Sundar et al. (2012) reported their hybrid ACO algorithm for MDS runs on average 137.99, 22.11 and 147.02 seconds

Table 2. Computational results with TS on the Genmax instances for MBV and MDS.

		MBV			MDS		
Instances		BEST	TS		BEST	TS	
n	d	UB	UB	CPU	UB	UB	CPU
20	1.5	1.40	1.40	1.54	5.20	5.20	1.91
20	2	0.60	0.60	1.24	2.00	2.00	1.70
20	4	0.00	0.00	1.03	0.00	0.00	1.36
20	10	0.00	0.00	0.98	0.00	0.00	1.35
20	15	0.00	0.00	0.83	0.00	0.00	1.00
30	1.5	2.40	2.40	2.60	9.20	9.20	3.59
30	2	0.60	0.60	2.34	2.60	2.60	3.09
30	4	0.40	0.40	1.98	1.80	1.80	2.81
30	10	0.00	0.00	1.58	0.00	0.00	1.74
30	15	0.00	0.00	1.24	0.00	0.00	1.64
40	1.5	2.00	2.00	3.12	8.20	8.20	3.56
40	2	0.60	0.60	2.45	1.80	1.80	3.28
40	4	0.00	0.00	2.18	0.00	0.00	2.46
40	10	0.00	0.00	1.86	0.00	0.00	2.08
40	15	0.00	0.00	1.23	0.00	0.00	1.33
50	1.5	3.20	3.20	6.84	14.20	14.20	8.76
50	2	0.80	0.80	5.98	4.00	4.00	7.30
50	4	0.00	0.00	4.78	0.00	0.00	6.41
50	10	0.00	0.00	4.21	0.00	0.00	5.18
50	15	0.00	0.00	3.86	0.00	0.00	4.63
100	1.5	6.00	5.80	21.5	28.80	28.60	30.75
100	2	2.00	2.00	18.6	11.20	11.00	23.44
100	4	0.00	0.00	17.6	0.00	0.00	19.89
100	10	0.00	0.00	14.8	0.00	0.00	18.94
100	15	0.00	0.00	14.2	0.00	0.00	16.19
300	1.5	20.40	20.20	146.3	99.20	96.80	188.73
300	2	6.80	6.60	138.5	40.20	40.00	185.59
300	4	0.20	0.00	127.3	0.00	0.00	156.58
300	10	0.00	0.00	109.3	0.00	0.00	134.44
300	15	0.00	0.00	97.6	0.00	0.00	111.26
500	1.5	33.40	33.20	657.3	164.80	162.20	893.93
500	2	10.60	10.20	587.2	66.80	64.20	675.28
500	4	0.60	0.60	512.3	1.00	1.00	589.15
500	10	0.00	0.00	498.5	0.00	0.00	618.14
500	15	0.00	0.00	476.3	0.00	0.00	509.64
1000	1.5	72.20	70.80	1438.3	352.80	348.40	1912.94
1000	2	24.60	24.20	1234.1	151.20	148.40	1838.81
1000	4	1.00	1.00	1005.4	2.60	2.60	1427.67
1000	10	0.00	0.00	986.1	0.00	0.00	1252.35
1000	15	0.00	0.00	947.3	0.00	0.00	1222.02
AVERAGE		4.75	4.67	227.51	24.19	23.81	297.27

Table 3. Computational results with TS on the Netgen instances for MBV and MDS.

		MBV			MDS		
Instances		BEST	TS		BEST	TS	
n	d	UB	UB	CPU	UB	UB	CPU
20	1.5	0.00	0.00	0.21	0.00	0.00	0.21
20	2	0.00	0.00	0.28	0.00	0.00	0.24
20	4	0.00	0.00	0.26	0.00	0.00	0.27
20	10	0.00	0.00	0.14	0.00	0.00	0.13
20	15	0.00	0.00	0.13	0.00	0.00	0.11
30	1.5	0.00	0.00	0.51	0.00	0.00	0.66
30	2	0.00	0.00	0.52	0.00	0.00	0.56
30	4	0.00	0.00	0.44	0.00	0.00	0.43
30	10	0.00	0.00	0.45	0.00	0.00	0.48
30	15	0.00	0.00	0.37	0.00	0.00	0.31
40	1.5	0.00	0.00	0.83	0.00	0.00	1.00
40	2	0.00	0.00	0.78	0.00	0.00	0.81
40	4	0.00	0.00	0.69	0.00	0.00	0.62
40	10	0.00	0.00	0.52	0.00	0.00	0.66
40	15	0.00	0.00	0.61	0.00	0.00	0.70
50	1.5	0.00	0.00	1.25	0.00	0.00	1.33
50	2	0.00	0.00	0.97	0.00	0.00	1.11
50	4	0.00	0.00	0.84	0.00	0.00	0.82
50	10	0.00	0.00	0.71	0.00	0.00	0.89
50	15	0.00	0.00	0.72	0.00	0.00	0.71
100	1.5	0.00	0.00	2.63	0.00	0.00	2.81
100	2	0.00	0.00	2.46	0.00	0.00	2.88
100	4	0.00	0.00	1.87	0.00	0.00	2.41
100	10	0.00	0.00	1.45	0.00	0.00	1.64
100	15	0.00	0.00	1.12	0.00	0.00	1.14
300	1.5	0.00	0.00	14.24	0.00	0.00	19.17
300	2	0.00	0.00	14.37	0.00	0.00	16.59
300	4	0.00	0.00	12.32	0.00	0.00	13.65
300	10	0.00	0.00	9.58	0.00	0.00	12.83
300	15	0.00	0.00	8.66	0.00	0.00	9.37
500	1.5	0.40	0.00	42.58	0.60	0.00	58.65
500	2	0.20	0.00	40.21	0.00	0.00	41.81
500	4	0.00	0.00	35.68	0.00	0.00	43.08
500	10	0.00	0.00	28.67	0.00	0.00	38.32
500	15	0.00	0.00	16.36	0.00	0.00	19.40
1000	1.5	1.40	0.80	254.18	3.00	0.00	317.63
1000	2	0.60	0.20	243.59	0.60	0.00	282.46
1000	4	0.00	0.00	198.34	0.00	0.00	218.13
1000	10	0.00	0.00	124.52	0.00	0.00	134.46
1000	15	0.00	0.00	91.36	0.00	0.00	115.95
AVERAGE		0.07	0.03	28.89	0.11	0.00	34.11

for Genmax, Netgen and Random classes, respectively. When comparing to the computation times spent by the TS heuristic for MBV on Genmax, Netgen and Random classes, i.e. 227.51, 28.89 and 326.69 seconds, and CPU time required by the TS heuristic for MDS on Genmax, Netgen and Random classes, i.e. 297.27, 34.11 and 442.58 seconds, respectively. We conclude that the proposed TS heuristic yields more accurate solutions at the expense of higher computational time.

4. CONCLUSION

We have developed an attribute based TS heuristic approach for both problems using new neighborhoods. Compared to recent hybrid ACO algorithms, the proposed TS heuristic for both MBV and MDS, yields high quality results. Finally, we should remark that the design of efficient exact algorithms remains as further research.

Table 4. Computational results with TS on the Random instances for MBV and MDS.

		MBV			MDS		
Instances		BEST	TS		BEST	TS	
n	d	UB	UB	CPU	UB	UB	CPU
20	1.5	1.00	1.00	2.1	3.40	3.40	3.00
20	2	0.00	0.00	1.5	0.00	0.00	1.85
20	4	0.00	0.00	1.1	0.00	0.00	1.66
30	1.5	2.60	2.60	4.3	10.20	10.20	5.33
30	2	0.20	0.20	4.1	0.60	0.00	5.04
30	4	0.00	0.00	3.5	0.00	0.00	4.48
30	10	0.00	0.00	3.1	0.00	0.00	4.62
40	1.5	3.20	3.20	9.6	13.20	13.20	12.29
40	2	1.20	1.20	8.3	4.80	4.80	12.12
40	4	0.00	0.00	8.4	0.00	0.00	10.75
40	10	0.00	0.00	7.6	0.00	0.00	9.35
40	15	0.00	0.00	7.1	0.00	0.00	8.52
50	1.5	3.00	3.00	12.3	13.40	13.40	15.38
50	2	1.00	1.00	10.5	3.60	3.60	15.65
50	4	0.00	0.00	9.8	0.00	0.00	13.52
50	10	0.00	0.00	7.5	0.00	0.00	11.63
50	15	0.00	0.00	6.2	0.00	0.00	9.55
100	1.5	6.60	6.60	87.3	30.80	30.60	104.76
100	2	2.20	2.00	81.3	11.40	11.20	113.01
100	4	0.00	0.00	76.5	0.00	0.00	110.16
100	10	0.00	0.00	64.8	0.00	0.00	87.48
100	15	0.00	0.00	51.2	0.00	0.00	58.37
300	1.5	20.20	19.80	657.3	96.20	93.80	874.21
300	2	6.60	6.00	598.14	37.20	36.00	735.71
300	4	0.00	0.00	475.3	0.00	0.00	598.88
300	10	0.00	0.00	368.1	0.00	0.00	555.83
300	15	0.00	0.00	312.3	0.00	0.00	443.47
500	1.5	32.60	31.20	935.3	161.60	159.60	1487.13
500	2	11.80	11.00	857.6	73.00	70.20	1294.98
500	4	0.60	0.40	798.5	0.60	0.40	1165.81
500	10	0.00	0.00	658.5	0.00	0.00	928.49
500	15	0.00	0.00	649.3	0.00	0.00	883.05
1000	1.5	70.20	69.20	1554.2	337.80	332.40	1787.33
1000	2	23.20	21.80	1124.3	142.40	139.20	1540.29
1000	4	0.60	0.40	1005.3	0.60	0.60	1477.79
1000	10	0.00	0.00	856.5	0.00	0.00	1053.50
1000	15	0.00	0.00	768.9	0.00	0.00	930.37
AVERAGE		5.05	4.88	326.69	25.43	24.94	442.58

Acknowledgements: This research is supported by the Turkish Scientific and Technological Research Council Research Grants No: 107M462 and 109M139, and Galatasaray University Scientific Research Project Grant No: 14.402.003.

REFERENCES

Ahuja, R.K., Magnanti, T.L., and Orlin, J.B.(1993). *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

Carrabs F., Cerulli R., Gaudioso M., and Gentili, M. (2009) Lower and upper bounds for the spanning tree with minimum branch vertices. Computational Optimization and Applications, 56, 405–438.

Capone, A., Corti, D., Gianoli, L., and Sanso, D. (2012) An optimization framework for the energy management of carrier ethernet networks with multiple spanning trees, *Computer Networks*, 56, 3666–3681.

Cerulli, R., Gentili, M., and Iossa, A. (2009). Bounded-degree spanning tree problems: models and new algorithms, *Computational Optimization and Applications*, 42, 353–370.

Gargano, L., Hell, P., Stacho, L., and Vaccaro, U. (2002). Spanning trees with bounded number of branch vertices, *Lecture Notes in Computer Science*, vol. 2380, Springer Verlag, Berlin.

Glover, F., and Laguna, M. (1997). *Tabu Search*, Kluwer, Dordrecht.

Stern, T.E., and Bala, K. (1999). *Multiwavelenght Optical Networks: A Layered Approach*, Prentice-Hall, NJ.

Sundar, S., Singh, A., and Rossi, A. (2012). New heuristics for two bounded-degree spanning tree problems, *Information Sciences*, 195, 226–240.

Taillard, E.D. (1993). Parallel iterative search methods for vehicle routing mehtods, *Networks*, 23, 661–673.