Performance comparison of the AS and GTS algorithms for weighted maximum leaf spanning tree problem

Paweł Czaderna[1] and Konrad Wala[2]

Academy of Computer Science and Management, Bielsko-Biała, Poland
[1] e-mail: pawel.czaderna@gmail.com , [2] e-mail: kwa@ia.agh.edu.pl

Abstract. In the paper the combinatorial model of the weighted maximum leaf spanning tree problem of simple graph is presented. We give a detailed description of three procedures of the neighbour trees generation to the basis one which are exploited during the course of tree improving process realized by the classical simulated annealing and genetic local search algorithm. Numerical results of improving processes for randomly generated graphs are included.

1 Introduction

The weighted maximum leaf spanning tree problem (WMLSTP) is a known NP-hard (see [7]) combinatorial optimization problem (see [6]) concerned with finding the spanning tree of an undirected, connected graph G, such that the sum of the weights of leafs is maximum. This problem is a generalization of the maximum leaf spanning tree problem (MLSTP) first time investigated by Dijkstra [6]. WMLSTP problem has many applications in the real world such as transportation, communication, or location of some facilities like computers in the network (see figure 1 for an example) or VLSI lay-out (see [1], [3], [4], [6], [8]).

Given an undirected and connected graph $G = (V, E)$, where V denoted the set of vertices with $n = |V| > 0$ and E the set of edges with $|E| > n - 1$ and a real number $w(j)$ for each vertex $j \in V$ called the weight of vertex j, the WMLSTP is formally defined as finding a spanning tree T^* on G, such that

$$w(T^*) = \max \{ \sum_{j \in L(T)} w(j) : \quad T \text{ is a spanning tree in } G \}$$

is the maximum taken over all possible spanning trees of graph G, where $L(T) = \{ i \in V : |\delta_T(i)| = 1 \}$ is the set of leafs of the tree T, and for $i \in V$: $\delta_G(i)$ denotes a set of edges adjacent to i in graph G, $\delta_T(i)$ is a set of edges adjacent to i in the spanning tree $T = (V, E(T))$, $E(T) \subset E(G)$, in G, $\delta_T(i) \subseteq \delta_G(i)$, the vertex i with $|\delta_T(i)| = 1$ is called a leaf (degree-one vertex) of tree T.

The difference between problem WMLSTP and MLSTP, where the problem is to maximize the leaf's number $|L(T)| \rightarrow \max$, is displayed in figure 2. For graph from Fig. 2a) the spanning tree with maximal weight of leafs equal 8 is displayed in Fig. 2c) whereas the tree in Fig. 2b) has the weight of leafs equal 0 in spite of greater number of leafs than tree – Fig.2c).

The paper is organized as follows: In section 2, the procedures for neighborhood tree generation are presented and in section 3, the optimization algorithms are outlined. Section 4 contains computational results concerning performance of the AS and GTS algorithms. In the last section the paper is concluded.

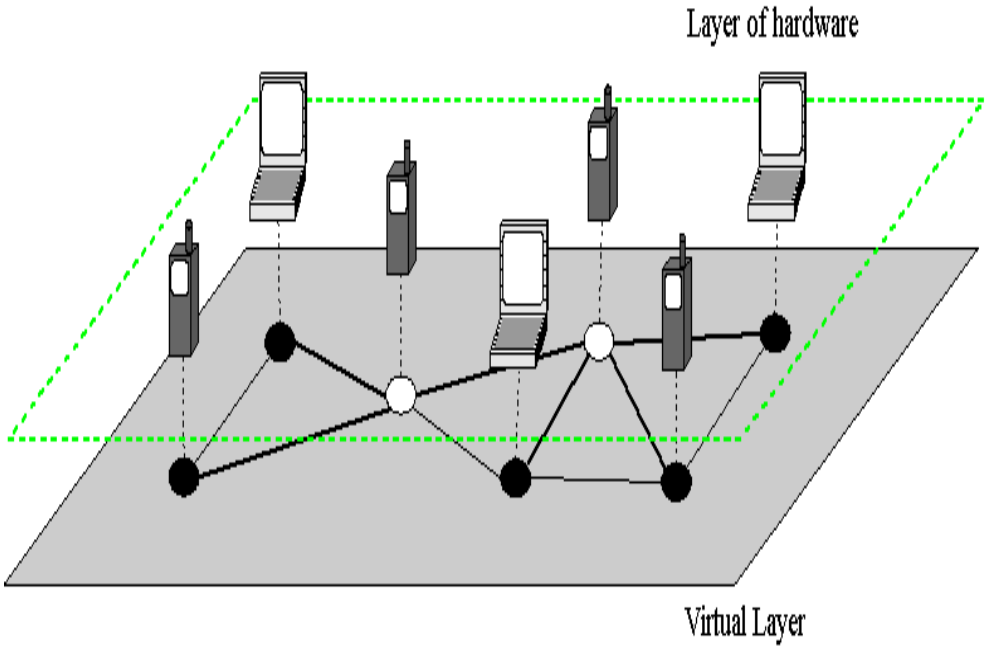


Figure 1. Example of the computer network.

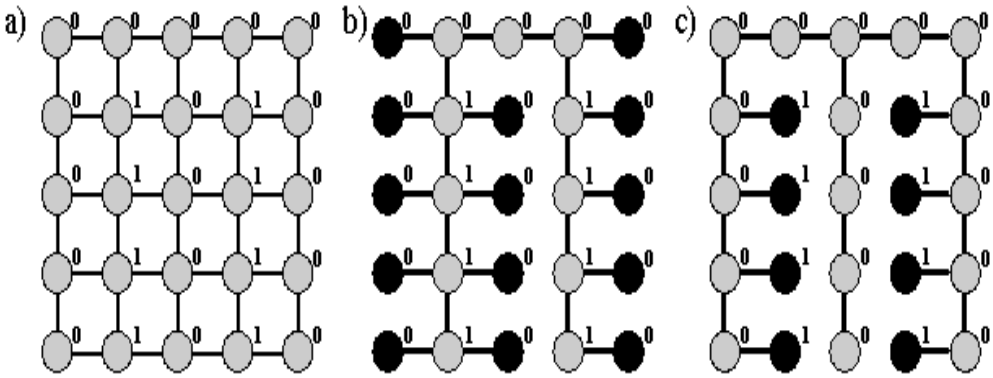


Figure 2. The example of the WMLSTP problem solution.

2 Procedures for neighbour trees generation

The essential problem of the search process for better solution in neighborhood of the basic tree is the way and cost of neighbour trees generation. One can apply "simple and cheep" modification rules of the basic tree and bear the expenses of verification of the generated

structures as well as rejection or reparation of infeasible one. We chose the method of neighbor tree generation, which consists in rejection some edges from the basic tree and than inserting new in more costly but controlled way. In presented below procedures, the generation process of neigbour trees is as follows: (*i*) randomly chosen edge or edges are removed from the basic tree, (*ii*) obtained connected components $T_1, T_2, ..., T_k$ of the basic tree are determined by the classical DFS (depth-first search) algorithm of graph search, (*iii*) the components are jointed by the use of the selected edges into new spanning tree.

On the ground of literature study (e.g., [1], [7]) and performed tests, we propose three procedures of neighbor tree generation:

1-change procedure. Let $V' = \{i \in V: |\delta_T(i)| > 1\}$ is the set of basic tree T vertices which are not leafs. We calculate randomly, uniform probability distribution, a vertex i from set V', denotation used in the paper: $i = random\ (V')$, as well as, also randomly, edge $\{i, j\}$, where $j = random(\{k \in N(i): \{i, k\} \in \delta_T(i)\})$ and $N(i) = \{k \in V : \{i, k\} \in \delta_G(i)\}$ is the set of neighbour vertices of the vertex i. After removing the edge $\{i, j\}$ from the basic tree one get two disconnected components T_1 and T_2 of the basic tree T where identification of the components vertices $V(T_1)$, $V(T_2)$, is performed by DFS algorithm in time $O(n+m_T)$. As the number of tree edges equals $m_T = n-1$ thus the complexity of DFS is $O(n)$. Next, the procedure tests the spanning trees obtained by inserting, on the place of removed edge $\{i, j\}$, edges $\{i, v\}$ where $i \in V(T_1)$, $v \in V(T_2)$, $v \neq j$, and finally returns the best tree, i.e. the tree of maximal leaf weight.

1-exchange procedure. Like in *1-change procedure*, the procedure removes randomly determined edge $\{i, j\}$ in time $O(n)$ and get two disconnected sets of vertices $V(T_1)$ and $V(T_2)$. Next, the set of edges $U = \{\{v, \mu\}: (v \in V(T_1)) \wedge (\mu \in V(T_2)) \wedge (\{v, \mu\} \in E)\}$ which can connect the components T_1 and T_2 into one tree is determined and then the procedure tests the spanning trees obtained by inserting, on the place of removed edge $\{i, j\}$, successively the edges from the set U and finally, it returns the best spanning tree.

k-change procedure. Let $V' = \{i \in V: |\delta_T(i)| > 1\}$ is the set of basic tree T vertices which are not leafs. Randomly, we determine the vertex $i = random(V')$ and than removing all edges of the set $\delta_T(i)$ we get the connected components $T_1, T_2, ..., T_k$, $2 \le k \le n$, of the basic tree where the identification of the vertices of the components $V(T_1), V(T_2), ..., V(T_k)$ is executed in time $O(n)$ by the DFS algorithm. Next, in time $O(n)$, for $\mu = 1, 2, ..., k$, there are randomly determine edges $\{i, j_\mu\}$ connected the isolated vertex i with components $T_1, ..., T_\mu, ..., T_k$ in neighbour tree where $j_\mu = random\ (V(T_\mu))$.

Let us notice that in case of equality $\delta_T(i) = \delta_G(i)$ for $i \in V'$ draw out by lot the 1-*change* and *k-change procedures* produce again the same basic tree what can often happen in case of sparse graph. In this case, it is worth to check the mentioned condition and break the computations.

3 Optimization algorithms

In this paper we investigate the simulated annealing – based heuristic called SA as well as genetic algorithm – based heuristic called GTS.

As a SA algorithm, we exploit classical one (see [2], [5]), described in figure 3, where the basic tree improvement trial consists in application with probability 1/3 one of the three function is to multiply temperature *temp* by some constant $\alpha < 1$. Using this reduction, the

SA algorithm works a constant m trials of every step with a fixed $temp$, and after m trials the temperature is reduced $temp := \alpha \times temp$.

1) Compute an initial spanning tree T (randomly or with the aid of constructive heuristic) , select parameters: m, α and an initial temperature $temp$.
2) SA algorithm:
   while $temp$ is not too close to 0 **do** {
   repeat m times {
   select a neighbour tree T_{neigh} (with the aid of application with probability 1/3 one of the three procedures of neighbour tree generation)
   if $w(T) \le w(T_{neigh})$ **then** $T := T_{neigh}$
   else generate a random number r uniformly in the range (0, 1);
   if $r < exp(-(w(T) - w(T_{neigh}))/temp)$ **then** $T := T_{neigh}$
   }
   $temp := \alpha \times temp$ (the cooling scheme);
   }

where: T : basic tree, $0.8 \le \alpha \le 0.99$, m : trail number in one iteration (number of trails with a fixed temperature $temp$)

Figure 3. Classical SA algorithm for WMLSTP problem.

As the GTS (Genetic Tabu Search) algorithm, we propose a modified genetic local search algorithm where each solution, individual of the population, generated by crossover procedure is improved by designed for WMLSTP randomized tabu search (RTS) instead of hill climbing algorithm. The population of GTS consists only of feasible solutions that is of spanning trees thus crossover operator and RTS algorithms have to generate only feasible descendents. The steps of GTS algorithm are following:

Step 1: Generation and evaluation of the spanning tree population.
Step 2: Drawing, by roulette mechanism, two parent trees, T_1 and T_2 , from the population.
Step 3: Generation the descendant tree T_3 of parents T_1 and T_2 by crossover operator.
Step 4: Improvement of the tree T_3 by RTS algorithm.
Step 5: The improved tree T_3 replaced the worst population tree.
Step 6: If the stop criterion is not fulfilled go to Step 2.

Crossover operator determines the genetic material of parents T_1 and T_2 as the set-theoretic sum of parents edges $E(T_1) \cup E(T_2)$ and then successively choose the edges from this set with probability proportional to the sum of edge incident vertices weight until successor T_3 is established(see figure 4 for an illustrative example).

Economize computation time; RTS algorithm exploits only *1-change* and *1-exchange* procedures for better solutions search process. Let us underline that both of these procedures have random components thus RTS algorithm does not locally improve the solutions but also play the role of mutation operator. Iteration of the RTS algorithm consists in generation new nieghbour spanning tree by means of one of 1-change or 1-exchange procedure chosen with ½ probability. The removed as well as the inserted edges, connected with generation new tree, get the status tabu for definite tabu tenure iteration number (see figure 5 for the illustrative example,

where the 1-*exchange* procedure generates, in one iteration, better new tree than the basic one). Obviously, in the process of removing and inserting edges realized by exploit procedures the tabu edges are not taken into consideration.

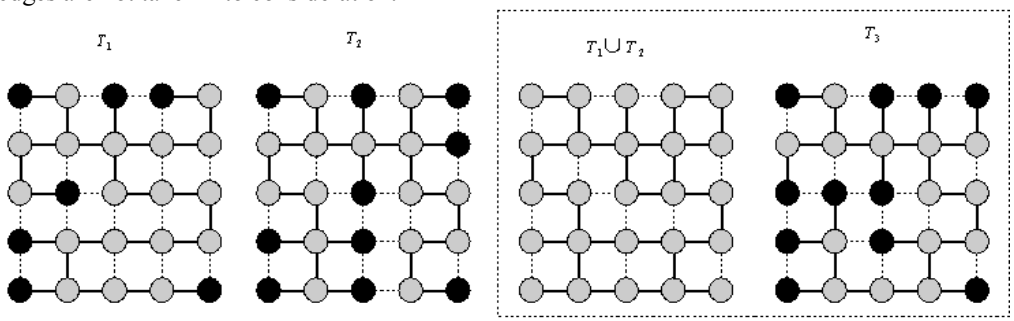
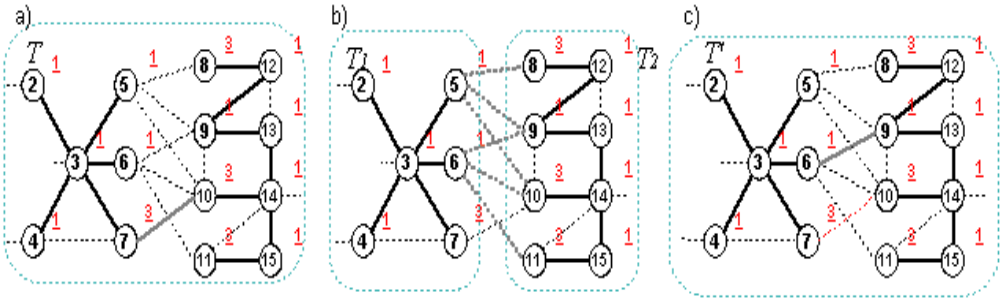


Figure 4. Example of the crossover process where $T_1 \square T_2$ stands for sum $E(T_1) \cup E(T_2)$.



We remove randomly edge {7, 10}

randomly.

We generate a set of the edges U = {{5, 8}, {5, 9}, {5, 10}, {6, 9}, {6, 10}, {6, 11}} and choose the best:
insert {5, 8}: $w(T) = 12$
insert {5, 9 }: $w(T) = 14$
insert {5, 10}: $w(T) = 12$
insert {6, 9 }: $w(T) = 15$
insert {6, 10}: $w(T) = 12$
insert {6, 11}: $w(T) = 10$

The best is $w(T) = 15$, so we at last insert edge {6, 9} to the tree and we have new spanning tree. Edges: {7, 10}, {6, 9} are tabu.

Figure 5. Illustration of an iteration of the RTS algorithm.

4 Computational results

The performance of the SA and GTS algorithms was evaluated on a large set of randomly generated test graphs for $n = 25, 50, 75, \ldots, 250$ vertices. For each n 50 test graphs (20 grid, 10 cubic, 20 with random connections in graphs) were generated; for each test 10 optimizations were performed. Before each series of computational experiment tests a lot of experimental work was done to adjust the algorithms parameters for particular graph vertices

number n well, where the main algorithms parameter, stop criterion, was established on the observation of the optimization process stagnation. During all computational experiments AS algorithm had the following constant parameters: initial temperature *temp* = 100, trial number

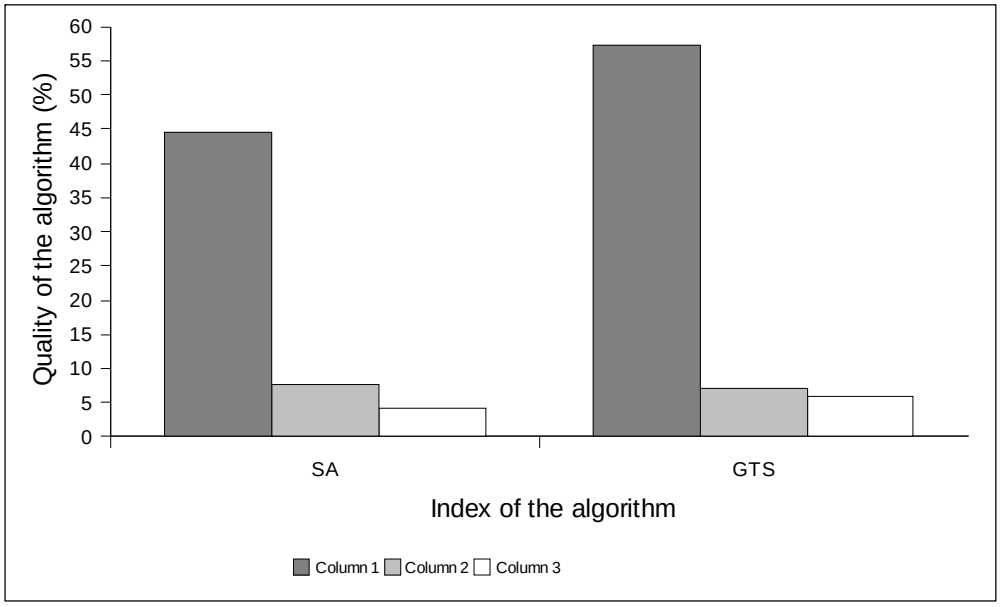


Figure 6. Computational results.

m = 20 and GTS algorithm had only constant population size equal 20 whereas all others algorithm's parameters were additionally tuning for each n.

Figure 6 and 7 display mean values of all series of computer experiments where one series consist of 10*50*10 = 5000 optimization processes. The first column of the Fig. 6 diagrams present the percentage improvement of the objective function realized by investigated SA and GTS algorithms in proportion to best solution in the randomly generated initial population and the second column in proportion to the best solution determined by 90 constructive heuristics. Besides, the third column presents the improvement index of the SA and GTS algorithms where this index was calculated as the mean value of improvement indices of test graphs determined as follows. Let us recall that for each test graph 10 optimizations was executed thus we calculated the mean value of objective functions of trees obtained during these optimizations and the improvement index of test graph equals the division of the objective functions mean value by objective function of the best spanning tree determined by 90 constructive heuristics for this graph.

Figure 7 presents the diagram of mean CPU time value, in seconds, of all series of numerical experiments for AS and GTS algorithms as the function of vertex number n. The numerical experiments were carried out on PC equipped with 1.5 GHz and 1 Gb processor.

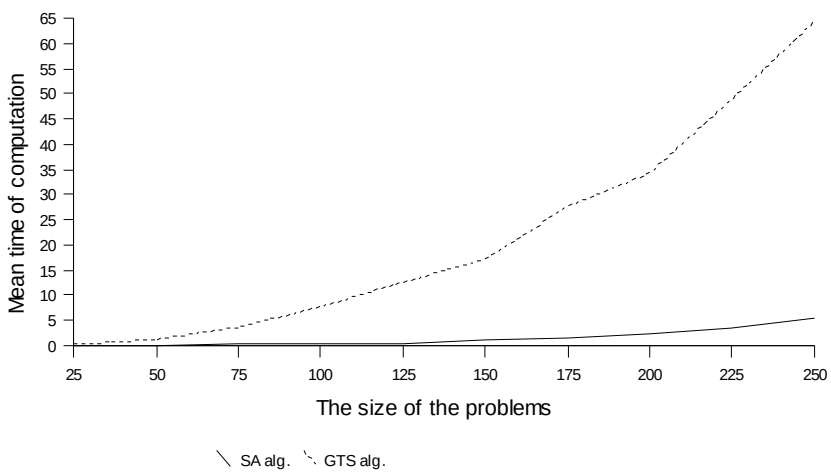


Figure 7. Mean time of computation

5 Conclusions

Finally, let us summarize our experience from the executed computational tests:

- The success of solution improvement algorithm mainly depends on careful tuning of algorithm's components and parameters.
- Algorithm's parameters are function of problem size - number of graph vertices n, e.g. we notice that the number iteration of RTS algorithm in GTS ought to be proportional to n.
- When tested on benchmark problem instances, SA algorithm equipped with proposed three neighbour tree generation procedures as well as GTS algorithm with RTS algorithm proved capable of achieving good results both time expense and optimum seeking.
- AS algorithm equipped with only two neighbour trees generation procedures computes unsatisfying solutions. Implementation in RTS algorithm all proposed neighbour trees generation procedures do not improve the GTS algorithm performance.

Bibliography

[1] Ahuja R.K., Orlin J.B., Sharma D., *Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem*. Working Paper, University of Florida, 2001. Submitted for publication.

[2] Czaderna P., Wala K., Simulated annealing in maximization of the leafs weight of spanning tree (in Polish). *Automatics No 143*, The Silesian University of Technology Press, Gliwice 2006.

[3] Fernandes L.M., Gouveia L., Minimal spanning trees with a constraint on the number of leaves. *European J. of Operational Research*, Vol.104, pp. 250-261, 1998.
[4] Fujie T., An exact algorithm for the maximum leaf spanning tree problem. *Computer & Operations Research 30*, 1931-1944, 2003.
[5] Kirkpatrick S., Gellat P.D., Vecchi M.P., Optimization by simulated annealing. *Science* 220 (1983), 671-680.
[6] Loryś K., Zwoźniak G., Approximation algorithms for Maximum-leaf Spanning Tree for cubic graphs. Mohring R. and Raman R. (Eds), *ESA 2002*, LNCS2461, pp. 686-698, Springer-Verlag 2002.
[7] Lu H., Ravi R., *The power of local optimization: Approximation algorithms for Maximum-leaf Spanning Tree - Proceedings of the Thirtieth Annual Allerton Conference on Communication, Control and Computing*, pp. 533-542, 1992.
[8] Lu H., Ravi R., A near-linear-time approximation algorithms for Maximum-leaf Spanning Tree. *Journal of Algorithms*, Vol. 29, No 1, pp. 132-141, 1995.