Beyond Unfeasibility: Strategic Oscillation for the Maximum Leaf Spanning Tree Problem

Jesús Sánchez-Oro(✉) and Abraham Duarte

Universidad Rey Juan Carlos, Madrid, Spain
{jesus.sanchezoro,abraham.duarte}@urjc.es

Abstract. Given an undirected and connected graph, the maximum leaf spanning tree problem consists in finding a spanning tree with as many leaves as possible. This $\mathcal{NP}$-hard problem has practical applications in telecommunication networks, circuit layouts, and other graph-theoretic problems. An interesting application appears in the context of broadcasting in telecommunication networks, where it is interesting to reduce the number of broadcasting computers in the network. These components are relatively expensive and therefore its is desirable to deploy as few of them as possible in the network. This optimization problem is equivalent to maximize the number of non-broadcasting computers. We present a strategic oscillation approach for solving the maximum leaf spanning tree problem. The results obtained by the proposed algorithm are compared with the best previous algorithm found in the literature, showing the superiority of our proposal.

Keywords: Telecommunication networks · Broadcasting · Spanning tree · Strategic oscillation

1 Introduction

Soft Computing is a discipline with the aim of solving optimization problems using a set of techniques such fuzzy logic, neural networks, or evolutionary computing, among others. Most of real world problems can be formulated as optimization problems. These real-life optimization problems are usually hard to solve because they have many local optima, making traditional methods to fall in local optima traps, from where is hard to escape. Besides exact methods, that can be only used for moderate size problems, there exists many approximation techniques for obtaining high quality solutions. Metaheuristics emerge as a set of methodologies which provide near optimal solutions in reasonable computing time. Some of the most used metaheuristics are GRASP [5,6], Tabu Search [3,15], Variable Neighborhood Search [17,21], and Scatter Search with Path Relinking [18,20], among others. Metaheuristics have been successfully applied to a large variety of optimization problems, such as the bipartite unconstrained 0–1 quadratic programming problem [3], the differential dispersion minimization problem [4], or the maximally diverse grouping problem [11].

© Springer International Publishing Switzerland 2015
J.M. Puerta et al. (Eds.): CAEPIA 2015, LNAI 9422, pp. 322–331, 2015.
DOI: 10.1007/978-3-319-24598-0_29

In this paper we focus on the maximum leaf spanning tree problem (MLSTP), which can be theoretically described as follows. Given an undirected and connected graph $G = (V, E)$, where V is the set of vertices and E the set of edges, the MLSTP consists of finding a spanning tree with as many leaves as possible. MLSTP can be trivially solved for complete graphs, since any spanning tree has the same number of leaves. However, it is $\mathcal{NP}$-hard for the general case [12]. Several approximation algorithms for the MLSTP have been proposed, being two approximation algorithms of factor 3 and 2 [19,22], respectively, the most relevant approaches. The problem has also been studied from an exact perspective [10] with an algorithm based on an original formulation previously proposed [7]. Finally, a new formulation and polyhedral investigation has been proposed in [9].

The MLSTP has practical applications in telecommunication networks, circuit layouts, and other graph-theoretic problems [23]. An interesting application for the MLSTP emerges in the context of broadcasting in telecommunication networks. Given a communication network, each computer in it must be able to transmit information to any other connected computer. Since not all computers are connected among them, it is necessary to add a special component in some computers of the network which transform them in broadcasting computers. This component permits the communication among those computers that are directly connected to it. Figure 1 shows an example with four different computers (A, B, C, and D), where there are three physical links ($A - D$, $B - D$, and $C - D$) represented with solid black lines. If we consider that computer D hosts a broadcasting component, then three new connections (represented with dotted lines) are created ($A - B$, $A - C$, and $B - C$).

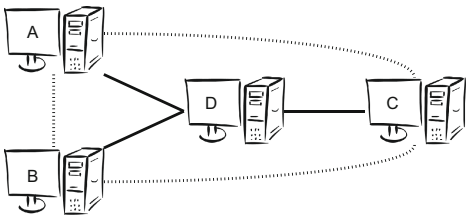
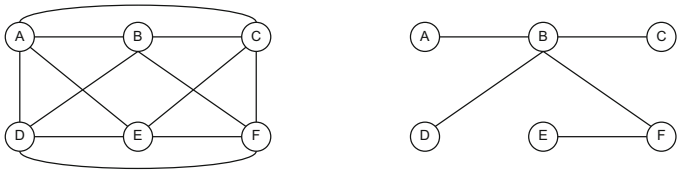


Fig. 1. The insertion of broadcasting computer D permits the communication between $A - B$, $A - C$ and $B - C$ (represented with dotted lines)

The price of these components is relatively expensive so it is important to reduce the number of broadcasting computers in a network in order to reduce its cost. This problem is equivalent to maximize the number of non-broadcasting computers in the network, which directly corresponds to MLSTP [8].

If we now analyze the problem from a theoretical point of view, the network can be represented with an undirected connected graph, where vertices and edges correspond to computers and physical links, respectively. If the graph is complete, then there is no need to add broadcasting computers. However, in the general case (sparse graphs), it is necessary to add one or more broadcasting

computers in order to communicate every pair of computers in the network. If we construct a spanning tree of the corresponding graph, we can see that hosting a broadcasting computer in each internal node, results in a fully communicated network. Then, the main objective is to reduce the number of internal nodes of the spanning tree, or equivalently, to maximize the number of leaves of the tree (i.e., the maximum leaf spanning tree problem).



(a) Example of a possible network represented as an undirected connected graph G

(b) Example of a spanning tree of G with root in B

Fig. 2. Construction of a spanning tree over a network in order to select the broadcasting nodes

Figure 2 shows an example of a network modeled as a graph (Fig. 2a), where the pairs of vertices that cannot directly communicate between them are (A, F) (B, E), and (C, D). An example of a spanning tree (Fig. 2b) is constructed starting from vertex B. Analyzing the spanning tree, it is necessary to add a broadcasting computer in each internal node of the tree (i.e., those which are not leaves, B and F). Then, the objective function of the spanning tree constructed in Fig. 2b for the MLSTP has a value of 4, since there are 4 leaves in the tree (vertices A, C, D, and E). More formally, given a graph G and being $\mathbb{T}(G)$ the set of all possible spanning trees of G, the MLSTP consists of finding a spanning tree $t^\star \in \mathbb{T}(G)$ with the maximum number of leaves. In mathematical terms,

$$t^\star \leftarrow \underset{t \in \mathbb{T}(G)}{\arg\max} |\{v \in V : \deg(v, t) = 1\}|.$$

where $\deg(v, t)$ represents the degree of the vertex v in tree t.

This problem is closely related to the Minimum Connected Dominating Set Problem (MCDSP) [16]. Specifically, a subset of vertices of a graph is a dominating set if each edge of the graph has, at least, one endpoint in it. The MCDSP consist in finding the connected dominating set of minimum cardinality. Notice that given an optimal connected dominating set for the MCDSP, we can construct a tree over it resulting in an optimal solution for the MLSTP and vice versa. The MCDSP was also proved to be $\mathcal{NP}$-hard in [12], and several approximation algorithms have been proposed [1,16]. Neither MLSTP nor MCDSP have been extensively studied from a heuristic point of view.

In this paper, we focus on the maximum leaf spanning tree problem, proposing a heuristic algorithm that follows a strategic oscillation approach (Sect. 2). The computational results and comparisons with previous algorithms are reported in Sect. 3. Finally, the conclusions derived from the research are sketched in Sect. 4.

2 Algorithmic Proposal

The algorithm proposed in this work follows the strategic oscillation methodology (SO) [11,13,14], which is closely related to tabu search. SO focuses the search in relation to a critical level, which is commonly a boundary in which an algorithm would normally stop. However, when reaching that boundary, SO modifies the rules of the search, allowing the algorithm to surpass it and continue the exploration. In the context of MLSTP, we select the feasibility of the solution as the boundary for strategic oscillation. Specifically, the SO algorithm proposed in this paper is intended to explore solutions beyond the feasibility frontier.

2.1 Solution Representation

Given a graph $G = (V, E)$, and a spanning tree T of G, a solution S for the maximum leaf spanning tree problem is represented by the subset of vertices in V, which contains the leaves of the tree. More formally,

$$S \leftarrow \{v \in V : \deg(v, T) = 1\}$$

Alternatively, we define the subset $S' \leftarrow V \setminus S$ as the one that contains those vertices which are internal nodes of T. For example, the solution S for the spanning tree represented in Fig. 2b is $S = \{A, C, D, E\}$, while $S' = \{B, F\}$. Notice that the objective function for the MLSTP, which is the number of leaves in T, is evaluated as $|S| = 4$.

We now define the graph $G' = (S', E')$, as the subgraph derived from G which contains the vertices in S' and the edges among them. In mathematical terms, the set of edges is defined as,

$$E' \leftarrow \{(u, v) \in E : u, v \in S'\}$$

The feasibility of a solution S is determined by two conditions. First of all, since S' represents the internal nodes of the spanning tree, the graph $G' = (S', E')$ must be connected. The second condition assures that every vertex in S is connected, at least, to one vertex in S', in order to assure that S contains the leaves of the tree. More formally,

$$\forall v \in S \quad \exists u \in S' : (u, v) \in E.$$

2.2 Constructive Procedure

The constructive procedure is intended to create a promising starting point for a search procedure. Instead of starting from a random solution, this procedure

tries to maximize the number of vertices in S, avoiding the exploration of low quality solutions. The constructive algorithm builds the corresponding solution in a greedy manner. The procedure starts by considering all vertices in S (i.e., $S = V$). The method then selects the vertex with the largest degree in G. It becomes the root of the spanning tree under construction and it is removed from S and inserted in S'. Then, a candidate list is created with the adjacent vertices of the root. The vertices in the candidate list are evaluated according to the greedy function, g, defined as:

$$g(v) \leftarrow |N(v) \cap S| - |N(v) \cap S'|$$

where $N(v) = \{u \in V : (v, u) \in E\}$ is the set of adjacent vertices to v.

In each iteration, the method selects the vertex $v^\star$ from the candidate list with the largest g-value updating accordingly sets S and S'. In the next iteration, the candidate list is updated with the adjacent vertices to $v^\star$ which belong to S. The method stops when the feasibility conditions defined in Sect. 2.1 are satisfied. Notice that the first feasibility condition is maintained during the construction, since the candidates are selected among the adjacency of vertices which are already in S'.

2.3 Strategic Oscillation

Before defining the strategic oscillation algorithm, it is necessary to define the type of moves used in the search. In particular, for the MLSTP, we propose two different moves. The first move, defined as $\text{drop}(v, S)$, consists of removing a vertex v from S, inserting it in S'. The second one, defined as $\text{add}(v, S)$, inserts the vertex v in the solution S, removing it from S'. In mathematical terms,

$$\text{drop}(v, S) = \begin{cases} S & \leftarrow S \setminus \{v\} \\ S' & \leftarrow S' \cup \{v\} \end{cases} \qquad \text{add}(v, S) = \begin{cases} S & \leftarrow S \cup \{v\} \\ S' & \leftarrow S' \setminus \{v\} \end{cases}$$

The drop-move never results in an unfeasible solution, since removing a vertex from S (i.e., a leaf of the spanning tree) does not break any feasible condition. Specifically, if we drop a vertex from S, all vertices in S remain connected to at least one vertex in S', and subgraph $G' = (S', E')$ remains connected. On the other hand, it is difficult to perform an add-move without obtaining an unfeasible solution. In particular, adding a vertex to S (and removing it from S') can eventually disconnect the subgraph G'. Furthermore, the second condition of feasibility is broken if there is any vertex in S whose only adjacent in S' is the added vertex. Notwithstanding, only add moves are able to improve the quality of a solution, since they increase the size of S, which determines the value of the objective function. In other words, adding vertices to S increases the number of leaves of the corresponding tree. However, dropping vertices from S would only reduce the number of leaves in the tree, and, therefore, the quality of the solution is deteriorated. For that reason, the strategic oscillation takes on special significance, since traditional improving methods would quickly find a basin of attraction from which is difficult to scape.

The main idea behind strategic oscillation is based on giving the algorithm the opportunity to explore unfeasible solutions and then bring them back to feasibility. Specifically, the search performed by the SO algorithm is based on adding some vertices to S with add moves (which leads to unfeasible solutions), and then repair the solution by dropping new vertices until the incumbent solution becomes feasible. Then, a solution is improved if and only if the number of added vertices is larger than the number of dropped ones. The number of vertices that are added in each iteration is determined by the parameter k, which indicates how far a solution is from feasibleness. Specifically, the oscillation strategy allows the procedure to visit solutions close to the feasibility region (small values of k), when it finds improvement moves, or far away from the feasibility region (large values of k) when no improvement is found.

Algorithm 1. Strategic Oscillation($G = (V, E), k_{\text{step}}, k_{\max}$)

1: $S \leftarrow \text{Construct}(G)$
2: $S' \leftarrow V \setminus S$
3: $k \leftarrow k_{\text{step}}$
4: **while** $k \leq k_{\max}$ **do**
5: $\quad \Delta^+ \leftarrow \emptyset$, $\Delta^- \leftarrow \emptyset$
6: $\quad$ **for** $i = 1$ **to** $k * |S'|$ **do**
7: $\quad\quad v \leftarrow \text{SelectRandom}(S')$
8: $\quad\quad \text{add}(v, S)$
9: $\quad\quad \Delta^+ \leftarrow \Delta^+ \cup \{v\}$
10: $\quad$ **end for**
11: $\quad$ **while not** feasible(S) **do**
12: $\quad\quad v^\star \leftarrow \arg\max_{v \in S} |N(v) \cap S| - |N(v) \cap S'|$
13: $\quad\quad \text{drop}(v^\star, S)$
14: $\quad\quad \Delta^- \leftarrow \Delta^- \cup \{v^\star\}$
15: $\quad$ **end while**
16: $\quad$ **if** $|\Delta^+| > |\Delta^-|$ **then**
17: $\quad\quad k \leftarrow k_{\text{step}}$
18: $\quad$ **else**
19: $\quad\quad k \leftarrow k + k_{\text{step}}$
20: $\quad\quad S \leftarrow S \cup \Delta^- \setminus \Delta^+$
21: $\quad\quad S' \leftarrow S' \cup \Delta^+ \setminus \Delta^-$
22: $\quad$ **end if**
23: **end while**

Algorithm 1 depicts the pseudocode of the strategic oscillation method proposed in this paper. It starts by constructing the initial solution with the greedy procedure described in Sect. 2.2 (step 1). SO is parametrized by $k_{\max}$ and k_{step}. The former indicates the maximum distance to feasibility that the algorithm is allowed to reach, while the latter represents the increment of that distance in each iteration. Starting from $k = k_{\text{step}}$ (step 3), SO iterates until reaching the maximum value of k (steps 4–23). For each iteration, the algorithm adds k vertices at random to the current solution (steps 6–10). At this step of the

algorithm, the solution has become unfeasible, so it needs to be repaired by dropping vertices until it becomes feasible again (steps 11–15). Notice that the selection of the next vertex to be dropped follows the same greedy criterion than the constructive procedure (step 12), which consists of selecting the vertex that maximizes the g-value (see Sect. 2.2). Finally, if the number of added vertices is lower than the number of dropped vertices, then an improved solution has been found, so the algorithm oscillates again closer to the feasibility by reducing k to the minimum value (step 17). On the other hand, if no improvement has been found, SO oscillates further from feasibility by increasing the value of k (step 19). In this case, the performed moves are undone by dropping the added vertices and vice versa (steps 20–21). The algorithm stops when reaching the furthest point from feasibility ($k_{\max}$) without finding an improvement, returning the best solution found.

3 Experimental Results

In this section we analyze the efficiency and effectiveness of the proposed algorithm when tackling the MLSTP. All the algorithms were coded in Java 8 and the experiments were conducted on an Intel Core i7 920 CPU (2.67 GHz) and 8GB RAM. The best previous work found in the literature [4] was originally proposed for solving a problem where the objective function is equivalent to minimize the number of internal nodes in a spanning tree, so the results can be directly adapted to the MLSTP [2].

In order to make a fair comparison, we have used in this work the publicly available instances[1] already used in previous works [2,4]. The instances correspond to 480 randomly generated networks. Each instance is generated based on two parameters: the number of nodes in the network (n) and the percentage of pairs which cannot share information in the network (p). Specifically, the values for n ranges from 40 to 500 and the values for p ranges from 10 % to 90 %. The computational experiments are divided into two parts: the preliminary experimentation and the final experimentation. The former is intended to select the best parameters for the proposed algorithm, and it uses a subset of 200 representative instances in order to avoid over-training, while the latter is devoted to compare the quality of our proposal against the best previous method [4]. The testbed is split into small instances ($k \leq 100$) and large instances ($k > 100$).

The preliminary experimentation is intended to set the best values for the parameters of the algorithm: k_{step} and $k_{\max}$. In particular, we performed a full-factorial design for a specific range of $k_{\text{step}} = \{0.01n, 0.05n, 0.10n\}$ and $k_{\max} = \{0.25n, 0.50n, 1.00n\}$ values, being n the number of vertices in the graph.

Table 1 shows the results obtained for the considered values of k_{step} and $k_{\max}$. The results are divided into small and large instances, reporting individual results for each n are shown, with the best value found highlighted in gray. The best results are consistently obtained when $k_{\max} = 1.00$. This value produces the furthest solutions from unfeasibility, which endorses the use of strategic

[1] http://homepage.univie.ac.at/ivana.ljubic/research/rlp/.

Table 1. Parameter tunning for the strategic oscillation algorithm over a subset of 200 representative instances. The best value for each n is highlighted in gray.

n		40		60		80		100	
$k_{\max}$	k_{step}	MLSTP	Time (s)	MLSTP	Time (s)	MLSTP	Time (s)	MLSTP	Time (s)
0.25	0.01	35.68	0.01	55.00	0.03	75.36	0.06	94.96	0.12
	0.05	35.63	0.01	54.96	0.01	75.36	0.02	94.92	0.04
	0.10	35.63	0.01	54.96	0.01	75.36	0.02	94.92	0.03
0.50	0.01	35.84	0.03	55.11	0.10	75.43	0.17	94.96	0.32
	0.05	35.68	0.01	55.19	0.03	75.46	0.05	94.96	0.09
	0.10	35.68	0.01	55.00	0.02	75.43	0.03	94.96	0.06
1.00	0.01	35.84	0.11	55.22	0.36	75.54	0.67	95.08	1.39
	0.05	35.79	0.02	55.19	0.07	75.46	0.13	95.04	0.29
	0.10	35.79	0.01	55.15	0.05	75.46	0.08	95.04	0.17

n		200		300		400		500	
$k_{\max}$	k_{step}	MLSTP	Time (s)	MLSTP	Time (s)	MLSTP	Time (s)	MLSTP	Time (s)
0.25	0.01	191.43	1.53	291.24	4.65	387.60	17.59	493.26	16.84
	0.05	191.43	0.72	291.24	2.36	387.60	9.80	493.22	8.80
	0.10	191.43	0.57	291.24	1.84	387.60	8.17	493.22	7.76
0.50	0.01	191.48	4.99	291.29	15.48	387.65	57.26	493.26	51.88
	0.05	191.43	1.46	291.26	4.72	387.65	18.86	493.22	16.10
	0.10	191.43	0.99	291.24	3.17	387.60	13.21	493.22	11.82
1.00	0.01	191.83	25.11	291.53	74.40	388.00	358.49	493.48	306.62
	0.05	191.65	4.96	291.41	15.53	387.75	61.14	493.26	62.89
	0.10	191.61	2.98	291.38	9.72	387.75	39.57	493.26	37.47

oscillation for the MLSTP. Among the variants with $k_{\max} = 1.00$, the best one is obtained when $k_{step} = 0.01$, which results in a fine-grained exploration of the solution space, since the distance from feasibility is increased in only 0.01 units in each iteration. As it can be seen in the results, the increment in the size of k_{step} results in a faster algorithm, but producing worse solutions. However, the computing time remains reasonable even for the lowest variant (which is the best one in quality), requiring on average less than one second per vertex in the graph. For that reason, we select $k_{step} = 0.01$ and $k_{\max} = 1.00$ for the comparison with the best previous method.

The final experiment is intended to compare the efficiency and effectiveness of the proposed algorithm when compared with the best previously published method, which corresponds with the GRASP algorithm originally proposed for the RLP [4]. The complete set of instances used in this work is not publicly available, so we have executed the same original code over the whole set of instances in the same computer, in order to provide a fair comparison.

Table 2 shows the comparison between the proposed SO algorithm and the previous best method (GRASP). Again, results from small and large instances are reported independently. Additionally to the number of vertices, in this case we add the value of p for each subset of instances. Notice that for a given number

Table 2. Comparison between SO and GRASP algorithms in small (left) and large (right) set of instances

		SO		GRASP	
n	p	MLSTP	Time (s)	MLSTP	Time (s)
10		38.60	0.02	38.60	0.19
20		38.00	0.03	38.00	0.22
30		38.00	0.04	38.00	0.26
40		37.40	0.06	37.60	0.29
40	50	37.00	0.07	37.00	0.31
60		36.40	0.09	36.60	0.32
70		35.70	0.12	35.70	0.34
80		34.40	0.14	34.40	0.36
90		30.90	0.24	30.50	0.43
10		78.10	0.13	78.10	1.64
20		78.00	0.18	78.00	2.07
30		77.40	0.30	77.70	2.55
40		77.00	0.37	77.00	2.74
80	50	76.60	0.52	76.80	2.92
60		76.00	0.69	76.00	3.18
70		74.70	0.97	75.00	3.58
80		73.00	1.44	73.00	3.88
90		68.30	1.91	68.80	4.74

		SO		GRASP	
n	p	MLSTP	Time (s)	MLSTP	Time (s)
10		58.10	0.06	58.10	0.68
20		58.00	0.09	58.00	0.83
30		58.00	0.13	58.00	0.95
40		37.40	0.06	37.60	0.29
60	50	57.00	0.23	57.00	1.17
60		56.00	0.27	56.00	1.21
70		55.00	0.36	55.20	1.32
80		53.20	0.46	53.60	1.48
90		49.60	0.87	49.60	1.76
10		98.00	0.23	98.00	3.21
20		98.00	0.37	98.00	3.83
30		97.20	0.53	97.30	4.66
40		97.00	0.73	97.00	5.71
100	50	96.00	0.94	96.10	6.21
60		95.20	1.25	96.00	6.54
70		94.20	1.76	94.50	7.20
80		92.60	2.76	92.60	7.95
90		87.60	3.74	88.00	10.13

		SO		GRASP	
n	p	MLSTP	Time (s)	MLSTP	Time (s)
10		198.00	1.77	198.00	39.92
30		197.00	4.40	197.00	55.67
200	50	196.00	9.23	196.00	69.94
70		193.10	15.75	193.00	96.54
90		184.70	52.11	184.10	153.83
10		398.00	11.51	398.00	424.57
30		396.90	35.06	397.00	618.54
400	50	395.00	71.14	395.00	808.05
70		392.00	152.25	391.40	1083.14
90		381.70	637.53	379.60	1809.32

		SO		GRASP	
n	p	MLSTP	Time (s)	MLSTP	Time (s)
10		298.00	5.08	298.00	152.44
30		297.00	14.94	297.00	228.06
300	50	295.10	28.12	295.00	314.53
70		292.70	64.22	292.40	369.05
90		282.60	176.29	281.70	599.69
10		498.00	21.81	498.00	843.94
30		496.20	70.56	496.90	1258.29
500	50	495.00	158.68	495.00	1665.78
70		492.00	375.99	491.00	2763.23
90		480.10	1456.21	478.00	6240.43

of vertices n, the higher the value of p, the harder the instance, so it is interesting to compare the evolution of both algorithms when considering harder instances. As stated in Table 1, the best results for each n and p are highlighted in gray. The first conclusion we can derive from these results is that the GRASP algorithm is much slower than the SO procedure. Specifically, the time needed by GRASP is six times larger on average than the time required by SO, and this difference increases with the number of vertices of the instance. The GRASP method performs slightly better in small instances, while the superiority of SO is clearly exposed in the set of large instances, showing the scalability of the proposed method when increasing the size and difficulty of the instance. We have conducted the Wilcoxon test to perform a pair-wise comparison between the proposed algorithm and the best previous method. The resulting p-value of 0.002 indicates that there are statistically significant differences between both methods, emerging SO as the best algorithm.

4 Conclusions

In this paper we have presented an algorithm based on the strategic oscillation methodology for solving the maximum leaf spanning tree problem. In particular, we have studied the effect of exploring unfeasible solutions and then bring them back to feasibility through the use of two move operators: add and drop. The algorithm has been compared with a previous method for a problem that can be adapted to the MLSTP over a set of 480 instances of different size and density. The computational results have shown that the proposed SO algorithm outperforms the previous method, based on GRASP methodology, both in quality and computing time. The scalability of the proposal versus the previous method has also been proved when increasing the size of the problems.

Acknowledgments. This research was partially supported by the Ministerio de Economía y Competitividad of Spain (Project Number TIN2012-35632-C02) and the Comunidad de Madrid (Project Number S2013/ICE-2894).

References

1. Butenko, S., Cheng, X., Du, D., Pardalos, P.M.: On the constructionof virtual-backbone for ad hoc wireless network. In: Butenko, S., Murphey, R., Pardalos, P.M. (eds.) Cooperative Control: Models,Applications and Algorithms, Cooperative Systems, vol. 1, pp. 43–54. Springer, US (2003)
2. Chen, S., Ljubić, I., Raghavan, S.: The regenerator location problem. Networks **55**(3), 205–220 (2010)
3. Duarte, A., Laguna, M., Martí, R., Sánchez-Oro, J.: Optimization procedures for the bipartite unconstrained 0–1 quadratic programming problem. Comput. Operat. Res. **51**, 123–129 (2014)
4. Duarte, A., Martí, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. Int. Trans. Oper. Res. **21**(4), 541–558 (2014)

5. Duarte, A., Sánchez-Oro, J., Resende, M., Glover, F., Mart, R.: Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. Inf. Sci. **296**, 46–60 (2015)
6. Feo, T.A., Resende, M., Smith, S.H.: A greedy randomized adaptive search procedure for maximum independent set. Oper. Res. **42**(5), 860–878 (1994)
7. Fernandes, L.M., Gouveia, L.: Minimal spanning trees with a constraint on the number of leaves. Eur. J. Oper. Res. **104**(1), 250–261 (1998)
8. Fernau, H., Kneis, J., Kratsch, D., Langer, A., Liedloff, M., Raible, D., Rossmanith, P.: An exact algorithm for the maximum leaf spanning tree problem. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 161–172. Springer, Heidelberg (2009)
9. Fujie, T.: The maximum-leaf spanning tree problem: Formulations and facets. Networks **43**(4), 212–223 (2004)
10. Fujie, T.: An exact algorithm for the maximum leaf spanning tree problem. Comput. Oper. Res. **30**(13), 1931–1944 (2003)
11. Gallego, M., Laguna, M., Martí, R., Duarte, A.: Tabu search with strategic oscillation for the maximally diverse grouping problem. J. Oper. Res. Soc. **64**(5), 724–734 (2013)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
13. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**(1), 156–166 (1977)
14. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell (1997)
15. Glover, F., Laguna, M.: Tabu search. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, pp. 2093–2229. Springer, US (1999)
16. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**(4), 374–387 (1998)
17. Hansen, P., Mladenović, N., Moreno, J.A.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
18. Laguna, M., Marti, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, Norwell (2002)
19. Lu, H., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithms **29**(1), 132–141 (1998)
20. Sánchez-Oro, J., Laguna, M., Duarte, A., Martí, R.: Scatter search for the profile minimization problem. Networks **65**(1), 10–21 (2015)
21. Sánchez-Oro, J., Pantrigo, J.J.: Duarte: Combining intensification and diversification strategies in vns. an application to the vertex separation problem. Comput. Oper. Res. **52**(Pt. B(0)), 209–219 (2014)
22. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)
23. Storer, J.: Constructing full spanning trees for cubic graphs. Inf. Process. Lett. **13**(1), 8–11 (1981)