Prog Artif Intell (2016) 5:121–128
DOI 10.1007/s13748-015-0076-7

REGULAR PAPER

Parallel strategic oscillation: an application to the maximum leaf spanning tree problem

Jesús Sánchez-Oro[1] · Borja Menéndez[1] · Eduardo G. Pardo[2] · Abraham Duarte[1]

Received: 9 December 2015 / Accepted: 23 December 2015 / Published online: 19 January 2016

Abstract The maximum leaf spanning tree problem consists in finding a spanning tree of a graph that maximizes the number of leaves that the tree has. This problem has been found to be $\mathcal{NP}$-hard for general graphs. It has several relevant applications in the context of telecommunication networks. In this paper, we tackle this problem by proposing the use of a parallel algorithm based on the strategic oscillation methodology. In particular, we propose two different parallel approaches and we compare our best variant with previous algorithms of the state of the art. The proposed approach outperforms previous ones in the state of the art, which is also confirmed by the use of statistical tests.

Keywords Telecommunication networks · Broadcasting · Spanning tree · Strategic oscillation

1 Introduction

Optimization is a key discipline in fields such as computer science, artificial intelligence, and operations research. Outside these scientific communities, the meaning of optimization becomes quite vague going to mean simply "do it as better as you can". In the context of this paper, the concept of optimization is conceived as the process of trying to find the best possible solution to an optimization problem, usually in a limited time horizon. There exists two different ways of tackling these problems. On the one hand, by considering them as a black-box problem, where the main available information is related with the solution representation; for instance, continuous [6], binary [19], or integer problems [21]. On the other hand, optimization problems can be approached by designing an specific algorithm. In this paper, we follow the second strategy. In particular, we solve maximum leaf spanning tree problem (MLSTP) using a parallel strategic oscillation procedure. This problem can be formally defined as follows. Let $G = (V, E)$ be an undirected and connected graph, where V is the set of vertices and E is the set of edges. MLSTP consists in finding a spanning tree of G with the maximum number of leaves and, therefore, the minimum number of internal nodes. This problem is trivial to solve for complete graphs, since every spanning tree presents the same number of leaves. However, it is proved to be $\mathcal{NP}$-hard for the general case [16].

More formally, the definition of the problem might be stated as follows: given a graph G and being $\mathbb{T}(G)$ the set of all possible spanning trees of G, the MLSTP consists in finding a spanning tree $t^* \in \mathbb{T}(G)$ with the maximum number of leaves. In mathematical terms,

$$t^\star \leftarrow \arg\max_{t \in \mathbb{T}(G)} |\{v \in V : |N_v(t)| = 1\}|$$

where $N_v(t)$ represents the set of adjacent vertices to v in the tree t.

In Fig. 1, we show an example of two particular graphs where the optimal solution is already known. Specifically,

✉ Abraham Duarte
abraham.duarte@urjc.es

Jesús Sánchez-Oro
jesus.sanchezoro@urjc.es

Borja Menéndez
borja.menendez@urjc.es

Eduardo G. Pardo
eduardo.pardo@upm.es

[1] Dpto. Informática y Estadística, Universidad Rey Juan Carlos, Madrid, Spain

[2] Dpto. Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, Spain

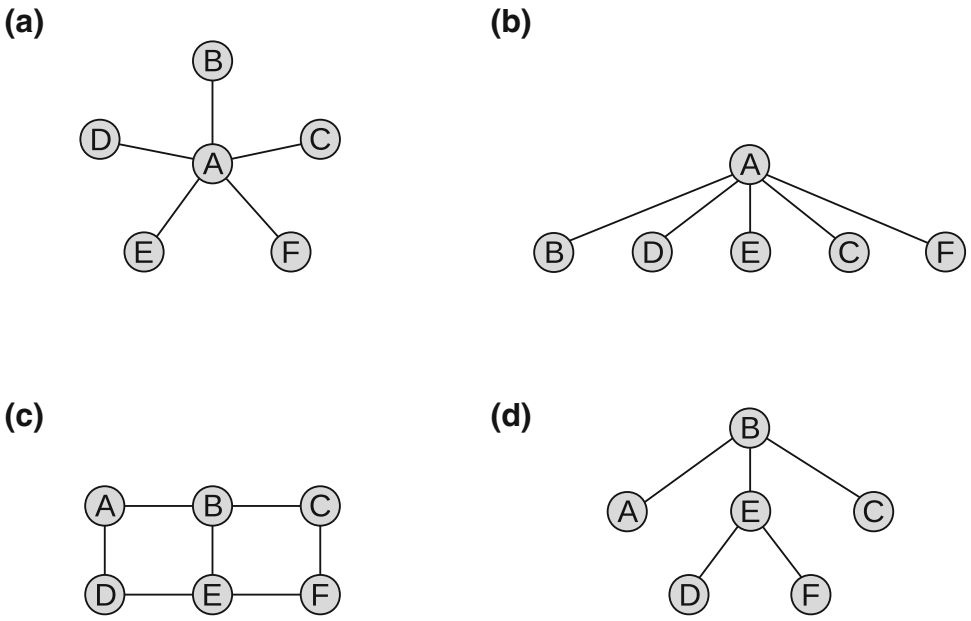

Fig. 1 Possible solutions derived from special graphs: star graph (**a**) and grid graph (**c**). **a** Example of a star graph G_{star} with six nodes. **b** Example of a spanning tree of G_{star} with root in A. **c** Example of a grid graph G_{grid} with six nodes. **d** Example of a spanning tree of G with root in B

in Fig. 1a, we present a star graph with six vertices and five edges. For this graph, the vertex A is connected with the rest of the vertices, while the rest of the vertices are only connected with A. In the particular case of star graphs the optimal value is always $|V| - 1$, being V the set of vertices of the graph, since all nodes of the tree can be arranged as leaves except one of them that will become the root of the tree. The resulting spanning tree associated to this graph is shown in Fig. 1b. As it was previously described, the node A will be the root of the tree.

Considering the aforementioned definition of the MLSTP, the spanning tree constructed in Fig. 1b has an objective function value of 5 since there are five leaves in the tree (vertices B, D, E, C, and F). Similarly, in Fig. 1c, we present an example of a grid graph with six vertices and seven edges. The optimal solution for this graph corresponds to the graph depicted in Fig. 1d. In this case, nodes B and E are internal, while A, D, F, and C are leaves. Therefore, the objective function value of this example is 4.

MLSTP has several practical applications, such as the design of ad hoc wireless networks, telecommunication networks, circuit layouts, and other graph-theoretic problems [28]. One of the most interesting applications of the MLSTP arises in the broadcast of information through a telecommunication network. In these networks, each computer is able to transfer information to any other computer connected to the network. Considering that not all the computers are directly connected one to each other, there are some of them that should work as broadcasters. A broadcaster is a computer that is able to allow communication among all the computers that are directly connected to it. To allow this, a special hardware component must be added to these computers. These components, that allow the communication between two computers with no direct link, are relatively expensive. Thus, it is crucial to reduce the number of broadcasting computers to reduce the cost of the network. This problem is equivalent to maximize the number of non-broadcasting computers in the network. Given a network, if we construct a spanning tree over it, then it is only necessary to convert in broadcasters those computers that are placed in internal nodes of the tree. Therefore, the larger the number of leaves in the tree, the smaller the number of broadcasters needed in the network. The problem of minimizing the number of internal nodes (i.e., minimize the number of broadcasting computers) is known in the literature as the regenerator location problem (RLP), which is completely equivalent to the minimum leaf spanning tree problem [10].

Another well-known application of the problem is related with the design of ad hoc wireless networks. These kind of networks generally have a topology that may change dynamically. For this type of application, it is desirable to set clusters of nodes such that they represent subgraphs of G connected among them with a small diameter. For each of these subgraphs, we must identify a *cluster-head*. Thus, the internal vertices of a maximum leaf spanning tree τ of G emerge as reasonable cluster-head candidates. It has to be specially considered when there exist internal vertices that share an edge of τ with a leaf vertex. In that case, the resulting cluster would contain not just the candidate cluster-head vertex, but also their associated leaf vertices. Furthermore, it may also contain some adjacent internal vertices of τ. For a more detailed description of this application we refer the reader to [2].

MLSTP has a close link to the minimum connected dominating set problem (MCDSP) [20], which has also been

proved to be $\mathcal{NP}$-hard [16]. Given a subset S of the vertices of a graph G, we say that S is a dominating set of G if every vertex in $V \backslash S$ has an adjacent in S. Thus, MCDSP is defined as finding a connected dominating set of maximum size. Hence, a set of non-leaves of a spanning tree is a connected dominating set and, similarly, a set of vertices outside a connected dominating set is a set of leaves of some spanning tree. Therefore, 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.

In this paper we center our attention on the maximum spanning tree problem. This problem has been approached from approximate, exact and heuristic perspectives. Among the former, several approximation algorithms have been proposed, where the most relevant ones presented an approximation of factor 2 and 3, respectively [22,27]. As far as the exact algorithms are concerned, the best approach [12] studies the problem with an algorithm based on the original formulation [9]. Polyhedral investigations were also conducted in [13] for the formulation previously used [12]. Finally, [3,4] formally introduce the RLP and present a branch-and-cut procedure for the Steiner arborescence problem with a unit degree constraint on the root node, which is shown to be equivalent to the RLP (and, consequently, equivalent to the MLSTP).

From a heuristic point of view, the first attempt to solve this problem was introduced in [3]. Specifically, the authors describe three greedy algorithms coupled with a local search procedure for the RLP. In [7], the authors presented new efficient implementations of the heuristics described in [3]. The authors additionally propose two new procedures: a GRASP and a biased random-key genetic algorithm, both for the RLP. More recently, Ref. [25] has proposed a method, specifically designated for the MLSTP, which explores both: feasible and unfeasible solutions. As fas as we know, this procedure currently obtains the best results in terms of quality and computing time.

The rest of the paper is organized as follows: in Sect. 2 we present a constructive procedure to generate good quality solutions for the MLSTP. In Sect. 3 we describe the strategic oscillation (SO) methodology used, while in Sect. 4 we present two novel parallel approaches based on this methodology. Computational results and comparisons with previous algorithms in the state of the art are shown in Sect. 5. Finally, we state our conclusions in Sect. 6.

2 Initial solution

A constructive procedure is a method intended to create a promising starting point for a search procedure, instead of starting from a random one. This kind of methods generally starts from scratch, returning a feasible solution. In this section, we propose a greedy strategy to generate a solution to the MLSTP. This initial solution will become the starting point for the improvement strategy (see Sect. 3).

Given a graph $G = (V, E)$, a solution of the MLSTP is a spanning tree T of G. Instead of working directly with the tree, we propose to represent the corresponding solution by means of the set of leaves $S \subset V$ of T. More formally,

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

The objective function for the MLSTP, which is the number of leaves in T, is evaluated as $|S|$. Additionally, let us denote as S' the subset that contains those vertices which are internal nodes of T. Therefore, $S \cup S' = V$ and $S \cap S' = \emptyset$.

The feasibility of a solution S is determined by two conditions: first, since S' represents the internal nodes of the spanning tree, the subgraph G' defined by the set of vertices S' and the set of edges $E' \leftarrow \{(u, v) \in E : u, v \in S'\}$ must be connected. Second, every vertex in S is connected, at least, to one vertex in S', to assure that S contains the leaves of the tree. More formally,

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

The constructive procedure proposed here tries to maximize the number of vertices in the initial solution S, building the corresponding solution in a greedy manner. The algorithm starts by considering all vertices in S (i.e., $S = V$). Then, the method selects the vertex with the largest degree in G which becomes the root of the spanning tree under construction. This vertex is removed from S and inserted in a new set S'. Again, we will use S' as the subset that will contain those vertices which are internal nodes of T. 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 the sets S and S' accordingly. 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 condition previously defined is satisfied.

3 Strategic oscillation

Strategic oscillation (SO) methodology was proposed in the context of tabu search [17,18]. The main idea behind this strategy relies on giving the algorithm the opportunity to

explore the search space after a critical level, which is commonly a boundary where an algorithm would normally stop [18]. Thus, it focuses on the search in relation to this critical level. An easy example of a reference critical level might be the set of feasible solutions. When reaching that boundary, SO would modify the rules of the search, allowing the algorithm to surpass it and to continue the search with the exploration through the set of unfeasible solutions. Later, if a promising unfeasible solution is found it will be necessary to bring it back to the feasible region of the search space. In the context of MLSTP, we select the feasibility criterion of the solution as the boundary for our strategic oscillation approach. Specifically, the SO algorithm proposed in this paper is intended to explore solutions beyond the feasibility frontier.

The SO method is based on two different move operators. The first one, denoted as drop(v, S), consists in removing a vertex $v \in S$, inserting it in S'; while the second one, add(v, S), inserts the vertex v in the solution S, removing it from S'. According to the problem definition, drop-move (applied to a feasible solution) does not ever result in unfeasible solutions, since removing a vertex from S (i.e., a leaf of the spanning tree) does not break any feasibility condition. Specifically, if we drop a vertex from S, the remaining vertices in S have, at least, one adjacent vertex in S'. Additionally, the subgraph $G' = (S', E')$ remains connected. On the other hand, it is difficult to perform an add-move to a feasible solution that does not produce 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 the unfeasible region and then bring the search back to the feasible region. Specifically, the search performed by the SO algorithm is based on removing some vertices from S with drop moves (which leads to unfeasible solutions), and then repair the solution by adding new vertices until the incumbent solution becomes feasible. Then, a solution is improved if and only if the number of added vertices is lower than the number of dropped ones. The number of vertices that are dropped 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: $S' \leftarrow V \backslash S$
3: $k \leftarrow k_{\text{step}}$
4: **while** $k \leq k_{\max}$ **do**
5: $\quad \Delta^- \leftarrow \emptyset$
6: $\quad \Delta^+ \leftarrow \emptyset$
7: $\quad$ **for** $i = 1$ **to** $k * |S|$ **do**
8: $\quad\quad v \leftarrow \text{SelectRandom}(S)$
9: $\quad\quad \text{drop}(v, S)$
10: $\quad\quad \Delta^- \leftarrow \Delta^- \cup \{v\}$
11: $\quad$ **end for**
12: $\quad$ **while not** feasible(S) **do**
13: $\quad\quad v^\star \leftarrow \arg\max_{v \in S'} |\{u \in S' : (u, v) \in E \wedge \nexists (u, w) \in E, w \in S \backslash \{v\}\}|$
14: $\quad\quad \text{add}(v^\star, S)$
15: $\quad\quad \Delta^+ \leftarrow \Delta^+ \cup \{v^\star\}$
16: $\quad$ **end while**
17: $\quad$ **if** $|\Delta^+| \leq |\Delta^-|$ **then**
18: $\quad\quad k \leftarrow k_{\text{step}}$
19: $\quad$ **else**
20: $\quad\quad k \leftarrow k + k_{\text{step}}$
21: $\quad\quad S \leftarrow S \cup \Delta^- \backslash \Delta^+$
22: $\quad\quad S' \leftarrow S' \cup \Delta^+ \backslash \Delta^-$
23: $\quad$ **end if**
24: **end while**

In Algorithm 1 we present pseudocode of the strategic oscillation procedure proposed in this paper. The method is parametrized by $k_{\max}$ and k_{step}. The former indicates the maximum distance to feasibility that the algorithm is allowed to reach (i.e., the number of extra vertices added), while the latter represents the increment of that distance in each iteration. The method starts by constructing an initial solution S (step 1) with the procedure proposed in Sect. 2. Remember that a solution to the MLSTP is a subset of vertices in V, which contains the leaves of the tree. Then, the rest of the vertices of the graph not assigned to the solution remains in S' and k is initialized to k_{step} (step 3). SO iterates until reaching the maximum value of k (steps 4–24). For each iteration, the algorithm moves k vertices at random from the set S' to the current solution S (steps 7–11). At this step, the solution might become unfeasible, so it needs to be repaired by dropping vertices until it becomes feasible again (steps 12–16). Notice that the selection of the next vertex to be dropped $v^\star$ follows a greedy criterion (step 13) where $N(v) = \{u \in V : (v, u) \in E\}$. Finally, if the number of added vertices is lower than the number of dropped vertices it means that an improved solution has been found, so the algorithm oscillates again closer to the feasibility by reducing k to the minimum value (step 18).

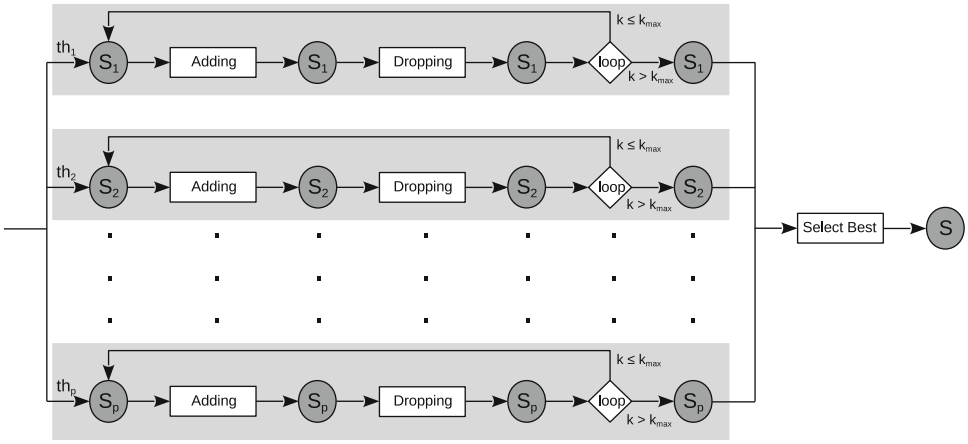

Fig. 2 Replicated independent SO scheme

On the other hand, if no improvement is found, SO oscillates further from feasibility by increasing the value of k (step 20). In this case, the performed moves are undone (steps 21–22). The algorithm stops when reaching the furthest point from feasibility ($k_{\max}$) without finding an improvement, returning the best solution found.

4 Parallel strategic oscillation

After decades of evolution in computer architecture, most of modern computers have several processors, enabling them to execute different programs simultaneously. Therefore, the programmers are now allowed to develop their algorithms following a parallel design to increase the performance of their programs. Parallelism is particularly effective in the case of heuristics and metaheuristics [1,29]. However, it is important to remark that algorithms must be redesigned to be adapted to a parallel architecture, since a direct parallel implementation of the original algorithm can eventually lead to low-quality results.

In this context, there exist several technologies for the implementation of parallel algorithms, such as threads, OpenMP, or CUDA. Some tutorials on parallel programming can be found in [5,14,24]. In this paper, we center our attention on threads. In programming languages, a thread is an independent flow of control executed in a processor. In this paradigm, *Pthreads* (POSIX threads) and *Java threads* are the most representative technologies. In particular, *Pthreads* were defined in the mid-90s as an effort to provide a unified set of *C* library routines to make multi-threaded programs portable. *Java threads* are a version of *Pthreads* for Java programming language and they offer the advantages of the portability inherent in Java programs. Moreover, Java threads can be easily used to tackle task parallel applications. Since all our algorithms have been implemented in Java, we select this technology to implement our parallel algorithms.

The aim of parallelism used in metaheuristics is usually to either reduce the computational time or to increase the exploration of the search space [15]. The first step in parallelization is to identify which parts of the sequential algorithm can be redesigned to be efficiently executed in parallel. There exist several examples of parallelization of different metaheuristics: genetic algorithms, ant colony optimization, scatter search, simulated annealing, tabu search, etc., (see [1] for a thoroughly review on parallel metaheuristics). One of the most recently parallelized methodologies has been variable neighborhood search [23]. In the last years, several parallel variants have been presented, such as synchronous parallel VNS (SPVNS), replicated parallel VNS (RPVNS), or replicated shaking VNS (RSVNS), among others [15].

In this paper we investigate an efficient parallelization of the SO methodology. Our parallel investigation is inspired by two of the previously mentioned algorithms: RPVNS and RSVNS. The idea behind RPVNS is to explore a wider portion of the solution space using a multi-start strategy, while RSVNS relies on a classical master–slave scheme where the master executes the main algorithm, and each slave executes, in parallel, a small part of the search. Since these strategies has lead to several successful research [8,26], we propose an adaptation of these algorithms to the strategic oscillation methodology.

The first parallel strategy, depicted in Fig. 2, is called replicated independent SO (RISO). The algorithm starts by creating p threads where each one executes an independent sequential strategic oscillation procedure. Therefore, in each thread a new solution is constructed, adding k vertices at random and, therefore, obtaining an unfeasible solution. Then, each thread removes vertices from its corresponding solution until obtaining a feasible one. Each thread stops when reaching the furthest point of unfeasibility ($k = k_{\max}$). The method stops when no improvement is found after executing a predefined number of independent iterations, which is a parameter of the algorithm. Finally, RISO returns the best solution found along the search.

The second novel parallel strategy for SO is called simultaneous exploration SO (SESO). In Fig. 3, we show shows

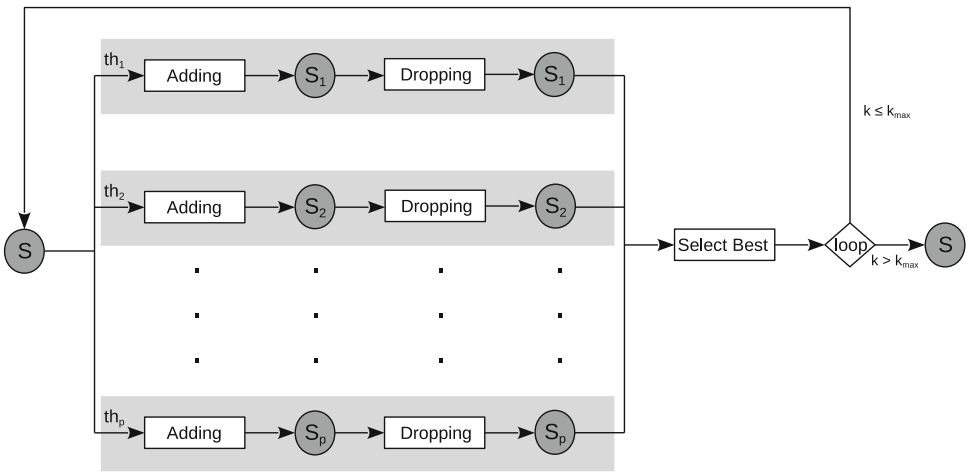

Fig. 3 Simultaneous exploration SO scheme

the proposed SESO, where the highlighted area is the part of the code executed in parallel by each thread. Specifically, the algorithm starts from an initial solution, S, generated by the constructive algorithm explained in Sect. 3. The algorithm then creates p threads to improve the original solution. In each thread, the solution is led to unfeasibility by adding nodes and then it is repaired by removing nodes until becoming feasible. At the end of each iteration, the algorithm waits with a barrier synchronization strategy until all threads have finished, updating the best solution found if necessary. The best solution is used as initial solution for the next iteration. It is important to remark that this strategy explores several unfeasible solutions derived from the original one (instead of exploring a unique unfeasible solution as in the sequential version). This feature allows the search to explore a wider portion of the solution space. For each iteration, if an improvement has been found in any of the threads, the distance to feasibility for the next iteration is reset ($k = 1$). Otherwise, it is increased ($k = k + 1$). Notice that the selection of the distance to unfeasibility is executed in the master thread (i.e., it is the only part of the SO procedure not executed in parallel).

The proposed parallel strategies present different main goals. On the one hand, RISO is intended to reduce the computing time by performing several iterations in parallel. On the other hand, SESO is devoted to explore a wider portion of the solution space, using the same or slightly more computing time. Therefore, the objective of RISO is to obtain the same quality in less computing time, while SESO tries to improve the quality of the algorithm when executing it for similar computing time.

5 Experimental results

In this section, we present the experiments performed to empirically study the influence of the proposed strategies in the quality of the solutions obtained. We then compare our best variant with the best algorithms identified in the state of the art. The best previous works found in the literature are the sequential strategic oscillation (SSO) [25] and a GRASP algorithm [7] (originally proposed for the RLP, which is equivalent to the MLSTP). We have implemented our algorithms in Java 8 and they were run on an Intel Core i7 920 CPU (2.67 GHz) and 8 GB of RAM.

We have considered three sets of instances publicly available to make a fair comparison. The first two sets (`small` and `large`) corresponds to 480 instances used in [7,25]. Each instance is generated by considering two parameters: the number of nodes in the network (n) and the percentage of pairs which cannot share information in the network (ρ). In particular, `small` instances sets $n = \{40, 60, 80, 100\}$ and $\rho = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$, while `large` instances consider $n = \{200, 300, 400, 500\}$ and $\rho = \{10, 30, 50, 70, 90\}$. We have additionally included a new set of instances, harwell-boeing, derived from the Harwell-Boeing Sparse Matrix Collection,[1] a dataset which has been used in several graph-related problems. This collection consists of a set of standard test matrices $M = M_{uv}$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The graphs are derived from these matrices by considering an edge (u, v) for every element $M_{uv} \neq 0$. From the original set we have selected a subset of 87 representative instances with n ranging from 30 to 685.

Computational experiments have been divided into two parts: the preliminary experimentation and the final experimentation. For each experiment, we report the following statistics: average number of leaves in the best solution, Avg.; average computing time in seconds, Time (s); average deviation with respect to the best solution found in the experiment, Dev (%); and number of best solutions found in

[1] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/.

Table 1 Performance of RISO when considering $p = \{2, 4, 8\}$

Algorithm	p	Avg.	Time (s)	Dev (%)	#Best
RISO	2	190.02	59.76	0.01	58
	4	190.02	45.15	0.01	58
	8	190.03	52.96	0.00	59
SESO	2	190.03	81.76	0.03	57
	4	190.07	93.90	0.02	59
	8	190.07	123.93	0.00	59

Table 2 Performance of RISO and SESO against GRASP and SSO

Algorithm	Avg.	Time (s)	Dev (%)	#Best
`Small`				
SSO	65.52	0.62	0.20	242
GRASP	65.62	2.95	0.09	271
RISO	65.54	0.28	0.17	247
SESO	65.55	1.20	0.15	251
`Large`				
SSO	342.96	134.35	0.03	187
GRASP	342.61	979.75	0.13	130
RISO	342.96	89.76	0.02	187
SESO	343.00	237.45	0.01	195
`Harwell-boeing`				
SSO	240.06	519.55	0.55	48
GRASP	233.71	652.37	4.35	34
RISO	240.87	550.76	0.30	56
SESO	240.71	643.99	0.29	62

the experiment, #Best. It is important to remark that, to avoid unacceptable computing times, we have set a maximum time limit of 1000 s for all the algorithms.

The preliminary experimentation is intended to test how the number of threads used in the parallel algorithms influences the quality of the search. For this experiment, we have selected a subset of 60 representative instances extracted from the `small` and `large` datasets to avoid over-training. Considering that the computer used for the experiments is able to execute 8 threads simultaneously, the parallel algorithms has been tested using $p = \{2, 4, 8\}$ threads.

In Table 1, we show the results obtained by RISO and SESO when considering $p = \{2, 4, 8\}$. It is important to remark that the results obtained by both methods are analyzed independently. Specifically, the deviation and number of best solutions for RISO are only considering the results of the three variants of RISO (similarly in SESO). Although there are no significant difference among the different configurations, it is worth mentioning that in both cases, the variant with the maximum number of threads is able to obtain slightly better results in terms of quality. Therefore, we select this configuration ($p = 8$) for the remaining experimentation.

The final experiment is devoted to compare the quality of the best parallel variants, RISO with eight threads and SESO with eight threads, against the best previous methods found in the literature: GRASP [7] and sequential SO (SSO) [25]. Table 2 presents the results divided by the considered set of instances: `small`, `large`, and `harwell-boeing`.

Analyzing the first testbed, `small`, we can see that all the algorithms perform similarly, although GRASP method is slightly better in terms of quality but requiring considerably more computing time. To test if GRASP is statistically better than the other methods, we have conducted the non-parametric Friedman statistical test [11], resulting in a p value lower than 0.001, which indicates that there are significant differences among the methods compared. Additionally, we have performed a non-parametric Wilcoxon signed rank test [30] between the two best methods (GRASP and SESO). The associated p value of 0.002 confirms again the superiority of the GRASP over our proposed method in this testbed.

If we now focus on the `large` dataset, we can clearly see the superiority of the parallel proposals in terms of deviation and number of best solutions found. In this case, the algorithm with the lowest quality is GRASP, which needs the largest computing time and obtains the lowest number of best solutions and the highest deviation. On the contrary, the parallel variants present the best quality. Specifically, RISO is able to outperform both SSO and GRASP by requiring half of the computing time used by SSO, and ten times faster than GRASP. In the case of SESO, the wider exploration of the solution space allows the method to obtain the best results but requiring larger computing time. In this case, the Friedman test returns a p value lower than 0.001, indicating that there are significant differences among the results.

Regarding the last testbed, `harwell-boeing`, the computing time of the compared algorithms are more or less the same, being SSO and RISO the fastest approaches. It is important to remark that the time limit of 1000 s is reached in several instances by all the algorithms although the instances present similar sizes than the other testbeds. This behavior can be partially explained because the instances of this testbed are a real challenge for heuristic algorithms in the context of MLSTP. The best method in terms of quality for this case is SESO, with the smallest deviation and the largest number of best solutions found. We confirm the superiority of our proposals by conducting a Friedman test. Specifically, the associated p value is lower than 0.001, while the average ranking of each method is 1.09 (SESO), 1.26 (RISO), 1.63 (SSO), and 2.02 (GRASP). Finally, we compare the best parallel method (SESO) with the best previous (sequential) method (SSO) with a Wilcoxon test, resulting in a p

value lower than 0.001, emerging SESO as the state-of-the-art method for the MLSTP.

6 Conclusions

In this work, two novel strategies for the parallelization of the strategic oscillation methodology are proposed. The strategies have been tested by applying them over different instances for the MLSTP. The experiments performed support the fact that the parallel proposal finds solutions of higher quality than the sequential one in less computing time. Furthermore, the results obtained by the proposed algorithms outperform the best previous methods found in the literature, becoming the state-of-the-art algorithms for the MLSTP. All results have been analyzed using non-parametric statistical test to validate the differences among the algorithms. The proposed parallel strategies have been designed to be easily adapted to any other sequential strategic oscillation approach, and can be also used to parallelize other traditional metaheuristics.

Acknowledgments This research has been partially supported by the Spanish "Ministerio de Economía y Competitividad", and by "Comunidad de Madrid" with Grants Refs. TIN2012-35632-C02 and S2013/ICE-2894, respectively.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms, vol. 47. Wiley, New York (2005)
2. Balasundaram, B., Butenko, S.: Graph domination, coloring and cliques in telecommunications. In: Handbook of Optimization in Telecommunications, pp. 865–890. Springer, New York (2006)
3. Chen, S., Ljubić, I., Raghavan, S.: The regenerator location problem. Networks **55**, 205–220 (2010)
4. Chen, S., Raghavan, S.: The regenerator location problem. In: Proceedings of the 2007 International Network Optimization Conference (INOC'07) (2007)
5. Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing), 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (2012)
6. Duarte, A., Martí, R., Gortázar, F.: Path relinking for large-scale global optimization. Soft Comput. **15**, 2257–2273 (2011)
7. Duarte, A., Martí, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. Int. Trans. Oper. Res. **21**, 541–558 (2014)
8. Duarte, A., Pantrigo, J.J., Pardo, E.G., Sánchez-Oro, J.: Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J. Manag. Math. **27**, 55–73 (2016)
9. Fernandes, L.M., Gouveia, L.: Minimal spanning trees with a constraint on the number of leaves. Eur. J. Oper. Res. **104**, 250–261 (1998)
10. 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: Parameterized and Exact Computation, pp. 161–172. Springer, New York (2009)
11. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc. **32**, 675–701 (1937)
12. Fujie, T.: An exact algorithm for the maximum leaf spanning tree problem. Comput. Oper. Res. **30**, 1931–1944 (2003)
13. Fujie, T.: The maximum-leaf spanning tree problem: formulations and facets. Networks **43**, 212–223 (2004)
14. Gao, G., Sato, M., Ayguadé, E.: Special issue on parallel programming with openmp. International Journal of Parallel Programming 36, (2008)
15. García-López, F., Melián-Batista, B., Moreno-Pérez, J., Moreno-Vega, J.: The parallel variable neighborhood search for the *p*-median problem. J. Heuristics **8**, 375–388 (2002)
16. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
17. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**, 156–166 (1977)
18. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell (1997)
19. Gortázar, F., Duarte, A., Laguna, M., Martí, R.: Black box scatter search for general classes of binary optimization problems. Comput. Oper. Res. **37**, 1977–1986 (2010)
20. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**, 374–387 (1998)
21. Laguna, M., Gortázar, F., Gallego, M., Duarte, A., Martí, R.: A black-box scatter search for optimization problems with integer variables. J. Glob. Optim. **58**, 497–516 (2014)
22. Lu, H.I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithms **29**, 132–141 (1998)
23. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
24. Oaks, S., Wong, H.: Java Threads. O'Reilly Media (2004)
25. Sánchez-Oro, J., Duarte, A.: Beyond Unfeasibility: Strategic Oscillation for the Maximum Leaf Spanning Tree Problem. In: Lecture Notes in Computer Science, vol. 9422. Springer, New York (2015)
26. Sánchez-Oro, J., Sevaux, M., Rossi, A., Martí, R., Duarte, A.: Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. Electron. Notes Discret. Math. **47**, 85–92 (2015)
27. Solis-Oba, R.: 2-Approximation Algorithm for Finding a Spanning Tree with Maximum Number of Leaves. Springer, New York (1998)
28. Storer, J.A.: Constructing full spanning trees for cubic graphs. Inf. Process. Lett. **13**, 8–11 (1981)
29. Talbi, E.G.: Metaheuristics: from design to implementation. Wiley, New York (2009)
30. Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. 1(6)80–83 (1945)