Restricted Swap-Based Neighborhood Search for the Minimum Connected Dominating Set Problem

Xinyun Wu and Zhipeng Lü
Laboratory of Smart Computing and Optimization, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, People's Republic of China

Philippe Galinier
Département de génie informatique et génie logiciel, École Polytechnique de Montréal, Canada

The minimum connected dominating set problem (MCDSP) has become increasingly important in recent years due to its applicability to mobile *ad hoc* networks and sensor grids. This paper presents a restricted swap-based neighborhood (RSN) tailored for solving MCDSP. This novel neighborhood structure is embedded into tabu Search (TS) and a perturbation mechanism is employed to enhance diversification. The proposed RSN-TS algorithm is tested on four sets of public benchmark instances widely used in the literature. The results demonstrate the efficacy of the proposed algorithm in terms of both solution quality and computational efficiency. In particular, the RSN-TS algorithm was able to improve the best known results on 41 out of the 97 problem instances while matching the best known results on all the remaining 56 instances. Furthermore, the article analyzes some key features of the proposed approach in order to identify its critical success factors. **© 2017 Wiley Periodicals, Inc. NETWORKS, Vol. 69(2), 222–236 2017**

Keywords: meta-heuristic; optimization; connected dominating set; swap-based neighborhood structure; tabu search; perturbation operator

1. INTRODUCTION

Given an undirected graph $G = (V, E)$, a set $X \subseteq V$ of vertices is said to be a dominating set if and only if any vertex not in X is adjacent to at least one vertex of X. In addition, a subset X of vertices is said to be connected if the subgraph induced by X is connected. Given an undirected graph, the objective of the minimum connected dominating set problem (MCDSP) consists of identifying a connected dominating set of minimum cardinality. This problem is NP-hard.

During the last decades, the MCDSP has received much attention due to its close connection with mobile ad-hoc networks [1]. In particular, the problem plays an important role in several applications, such as broadcast routing [8, 28, 31], power management [6, 9, 10], and fiber optical networks [7].

Due to the importance of the problem, many solution techniques have been developed for the MCDSP, in particular exact algorithms [14, 22, 27], approximation algorithms [16, 24, 29], and heuristics [5, 18, 25].

In this article, we propose a new neighborhood structure based on a restricted set of swaps, which is embedded into a tabu Search (TS) algorithm, for solving MCDSP. Salient features of the proposed algorithm are a fast incremental neighborhood evaluation technique and a diversification technique that relies on an adaptive perturbation mechanism.

The proposed RSN-TS algorithm is tested on four sets of 97 benchmark instances widely used in the literature. The experiments demonstrate the effectiveness and efficiency of our proposed algorithm. In particular, the RSN-TS algorithm is able to improve the best known results on 41 instances within very short computing times, while matching the best known results for all the remaining ones.

The following of the article is organized as follows. Preliminary definitions and a review of literature are presented in Section 2. The local search approach proposed to solve the problem is described in Section 3. The proposed algorithm is detailed in Section 4. Experimental results performed by using the proposed algorithm are described in Section 5. The analysis of several important features of the proposed RSN-TS algorithm is given in Section 6, before concluding the article in Section 7.

2. DEFINITIONS AND REVIEW OF LITERATURE

In this section, we first present some definitions related to connected graphs, articulation points, and dominance in

Received January 2016; revised December 2016; accepted December 2016
Correspondence to: Z. Lü; e-mail: zhipeng.lv@hust.edu.cn
Contract grant sponsor: National Natural Science Foundation of China; Contract grant number: 61370183
Contract grant sponsor: New Century Excellent Talents in University (NCET 2013)
DOI 10.1002/net.21728
Published online 18 January 2017 in Wiley Online Library (wileyonlinelibrary.com).

graphs. Then, we propose a short review of literature on the MCDSP in which we depict some applications of the problem and quote different exact algorithms, approximation algorithms, and heuristics proposed to solve the problem.

2.1. Preliminary Definitions

2.1.1. Connected Set of Vertices. Consider an undirected graph $G(V, E)$ and a set $X \subseteq V$ of vertices. The subgraph of G induced by X, denoted by $G[X]$, is the graph whose vertex set is X and whose edge set contains the edges in E having their two endpoints in X. In addition, we say that $X \subseteq V$ is a *connected set* of vertices if $G[X]$ is connected.

2.1.2. Articulation Points. For a connected undirected graph $G(V, E)$, we say that a vertex $v \in V$ is an *articulation* point of G if its removal would disconnect the graph. The set of articulation points of G will be denoted by $Art(G)$. This set can be computed in linear time, that is, in $O(|V| + |E|)$, by using the algorithm proposed by [17].

2.1.3. Dominating Set. Given an undirected graph $G(V, E)$ and a set $X \subseteq V$ of vertices, $\Gamma_G(X) = X \cup \{v \in V | u \in X, \{u, v\} \in E\}$ denotes the set of vertices which are in X or adjacent to a vertex in X. If $\Gamma_G(X) = V$, then X is said to be a *dominating set* of G.

2.1.4. MCDS and k-CDS Problems. Given a connected undirected graph $G(V, E)$, the goal of the *minimum CDS problem* (MCDSP) is to find a connected dominating set of G whose cardinality is minimum. The decision version of the MCDSP is named the *k -CDS problem*. Given a connected graph G and an integer k, the goal of this problem is to find a connected dominating set X of G whose cardinality is at most k, or to determine that such a set X does not exist. The k-CDS problem was proven to be NP-complete [13].

2.2. Review of Literature

During the last decades, MCDSP has received extensive attention due to its close connection with mobile *ad hoc* networks. The CDS can be applied to enhance the efficiency of the multicast/broadcast routing by alleviating the broadcast storm problem [8, 28, 31], which also plays an important role in power management [6, 9, 10]. The MCDSP plays an important role in the design of *ad hoc* wireless networks, where network topologies may change dynamically. A detailed discussion about this application can be found in [1]. Another application of MCDSP is the design of fiber optical networks, where information on regenerators may be required at some network vertices [7]. For a broader discussion of the CDS problem, please refer to [33].

The MCDSP was shown to be equivalent to a well-known problem named the maximum leaf spanning tree problem (MLSTP). For MLSTP, investigations are carried out in [12]. A branch-and-bound algorithm is proposed in [11], where insights lead to the efficient calculation of the corresponding linear programming relaxation bounds. Advanced heuristic methods for MLSTP are presented and discussed in [7, 22].

Many effective algorithms have been proposed for the MCDSP in the last decades. These algorithms can be classified into three categories: exact algorithms, approximation algorithms, and meta-heuristic algorithms.

Several exact algorithms have been proposed to solve the MCDSP in the literature. In particular, an integer programming formulation and new valid inequalities are presented in [27]. These authors also propose a branch-and-cut algorithm based on a reinforced formulation. This work has been extended by [14]. The decomposition algorithm proposed in this paper is the best performing exact algorithm for MCDSP.

With respect to approximation algorithms, two polynomial time algorithms for MCDSP that achieve approximation factors of $2H(\Delta) + 2$ and $H(\Delta) + 2$ are presented in [16], where Δ is the maximum degree and H is the harmonic function. Another approximation algorithm for MCDSP is presented in [29] with an approximation factor of at most 8 and a time complexity of $O(n)$, where n is the number of vertices. The collaborative cover heuristic proposed in [24] achieves the performance ratio of $(4.8 + \ln 5)opt + 1.2$, where opt is the size of the optimal CDS.

Several meta-heuristic algorithms have been proposed for tackling the MCDSP. Simple heuristics are presented in [23] for various unit disc graph related problems, including MCDSP. A greedy heuristic is proposed in [5]. This heuristic starts with a CDS consisting of all vertices. Then, it reduces the size of the current CDS by excluding some vertices one by one using a greedy criterion. A simulated annealing and a tabu search algorithm are proposed in [25]. A competitive one-step ant colony algorithm is proposed in [18], based on the approximation algorithm presented in [16]. Another variation of MCDSP which requires the dominating set to be a 2-connected graph is studied in [30].

Finally, while the above papers are mainly focused on the static mobile *ad hoc* networks environment, an algorithm which is also suitable for a dynamic environment is proposed in [19].

3. A LOCAL SEARCH APPROACH TO THE *k*-CDS PROBLEM

Our approach to tackle the MCDSP is based on a reduction to the decision version of the problem (the k-CDS problem). In order to solve a MCDSP instance, that is, to find a minimum CDS of the input graph G, our algorithm tries repeatedly to find in G a CDS of size k for decreasing value of k. That is to say, it tackles a series of k-CDS problem instances. More details will be given in Section 4.1.

In this section, we first present the local search approach to tackle the k-CDS problem, that is, we will define the search space, the cost function, and the neighborhood function. The procedure to tackle the k-CDS problem will be described in the next section.

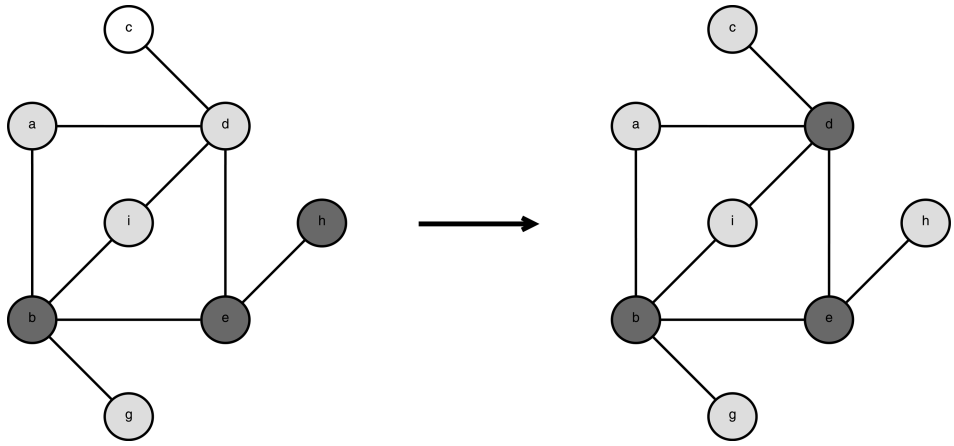

FIG. 1. Illustration of the restricted swap operator $< h, d >$. (a) v100_d70 (b) v200_d5 (c) ieee_57_bus (d) ieee_300_bus.

In the remainder of this section, we denote by $G = (V, E)$ the input graph, and by k the size of the solution. Recall that $G = (V, E)$ is an undirected connected graph.

3.1. Search Space and Cost Function

The search space (set of configurations) explored by our local search procedure is denoted by $\mathcal{S}$. In the proposed local search approach, a configuration X is any connected set of k vertices. Given a configuration $X \in \mathcal{S}$, we denote by $X^+ = \Gamma_G(X) \backslash X$ the set of vertices that do not belong to X and that are dominated by X, and by $X^- = V \backslash \Gamma_G(X)$ the set of vertices that are not dominated by X.

The cost $f(X)$ of configuration X is defined as the number of vertices that are not dominated by X: $f(X) = |X^-|$. One notices that a configuration X represents a solution of the k-CDS (i.e., a connected dominating set of size k) if and only if $f(X) = 0$.

3.2. Neighborhood Definition

Given a configuration $X \in \mathcal{S}$, we denote by $< x, y >$ the operation, named a swap, by which a vertex $x \in X$ is removed from X, while a new vertex $y \in V \backslash X$ is added into X. In addition, $X \oplus < x, y >$ denotes the configuration obtained by applying the swap $< x, y >$ to X. Thus, we have

$$X \oplus < x, y >= X \backslash \{x\} \cup \{y\} \quad (1)$$

However, as applying a swap may disconnect configuration X, the set of the moves applicable to X will be defined as a restricted set of swaps. This set, denoted by $\mathcal{M}(X)$, consists of the set of the swaps $< x, y >$ that satisfy the three following properties:

(P1) $x \in X \backslash Art(G[X])$;
(P2) $y \in X^+$; and
(P3) $\exists z \in X \backslash \{x\}, \{z, y\} \in E$.

According to property (P1), the vertex x to be removed from the configuration is any vertex of X that is not an articulation point of the subgraph induced by X. According to property (P2), the vertex y to be added into the configuration is any vertex in $V \backslash X$ that is connected to at least one vertex of X. According to (P3), x is not the only vertex in X to which y is connected.

We can notice that (P1) guarantees that $X \backslash \{x\}$ is connected and that (P2) guarantees that $X \cup \{y\}$ is connected. However, it is easy to see that (P1) and (P2) do not suffice to guarantee that $X \oplus < x, y >= X \cup \{y\} \backslash \{x\}$ is also connected. However, with the addition of property (P3), it is actually the case. As a result, applying a swap of $\mathcal{M}(X)$ to a configuration X produces a connected set of vertices. The proof of this property will be presented below in Theorem 1.

The definition of a move is exemplified in Figure 1, where $X = \{b, e, h\}$. The vertices in dark gray, light gray, and white represent the elements in X, $X^+ = \{a, d, i, g\}$, and $X^- = \{c\}$, respectively. Let us denote by $X' = X \oplus < h, d >$ the configuration obtained by applying move $< h, d >$ to configuration X. We have $X' = \{b, e, d\}$, $X'^+ = \{a, h, i, g, c\}$ and $X'^- = \emptyset$.

Theorem 1 states that a move defined according to the RSN preserves the connectivity of a set of vertices, and Theorem 2 shows that such a move transforms a configuration into a new configuration.

Theorem 1. *If X is connected and $< x, y > \in \mathcal{M}(X)$, then $X' = X \oplus < x, y >$ is connected.*

Proof. As $x \in X \backslash Art(G)$, we have that $X \backslash \{x\}$ is connected. Let us consider any two vertices u and v in X', such that $u \neq v$. There are two possible cases according to whether y is one of these two vertices or not. If $u \neq y$ and $v \neq y$ (*case 1*), as $X \backslash \{x\}$ is connected and $u, v \in X \backslash \{x\}$, there is a path between u and v in $G[X \backslash \{x\}]$. Thus, this path is also a path between u and v in $G[X \backslash \{x\} \cup \{y\}] = G[X']$. If $u = y$ (*case 2*), according to (P3), there exists a vertex $z \in X \backslash \{x\}$ such that y and z are adjacent. Also, as $z, v \in X \backslash \{x\}$, there is a path $z = v_1, \ldots, v_q = v$ between z and v in $G[X \backslash \{x\}]$. Therefore, $u = y, z = v_1, \ldots, v_q = v$ is a path between u and v in $G[X \backslash \{x\} \cup \{y\}] = G([X'])$. We have shown that in both cases 1 and 2, any two vertices u and v are linked by a path between u and v in $G[X']$. Thus, X' is connected. ∎

Theorem 2. *If $X \in \mathcal{S}$ and $< x, y > \in \mathcal{M}(X)$, then $X' = X \oplus < x, y > \in \mathcal{S}$.*

Proof. As $X \in \mathcal{S}$, we must have that $|X| = k$. Therefore, as $x \in X$ (P1) and $y \in V \backslash X$ (P2), we must have that $|X'| = |X \cup \{y\} \setminus \{x\}| = k$. Also, as $X \in \mathcal{S}$, X is connected. Thus, according to Theorem 1, X' must be connected. As $|X'| = k$ and X' is connected, we have $X' \in \mathcal{S}$. ∎

3.3. Fast Incremental Evaluation of RSN

Given a move $m \in \mathcal{M}(\mathcal{S})$, we denote by $\delta(m)$ the score of m, that is, its impact on the score of the configuration. Thus, $\delta(m)$ is defined according to

$$\delta(m) = f(X \oplus m) - f(X) \tag{2}$$

However, it will be time consuming to use this equation to calculate δ directly, because the time complexity of the calculation will be $O(|V|)$. In fact, we can evaluate δ by an auxiliary data structure in a fast manner. The search procedure maintains a list L, where element $L[i](1 \leq i \leq n)$ measures the number of vertices in X that are connected to vertex i.

$$L[i] = |\{j : \{i,j\} \in E, j \in X\}| \tag{3}$$

Thus, whether a vertex u is in $\Gamma_G(X)$ can be quickly determined by checking whether $L[u]$ is larger than 0. With this data structure, the score of $m =< x, y >$ can be calculated by:

$$\delta(m) = \delta^-(x) + \delta^+(y) \tag{4}$$

where $\delta^-(x)$ represents the changing value caused by removing x from X, while $\delta^+(y)$ the changing value caused by adding y into X. Formally,

$$\delta^-(x) = |\left\{v : v \in C_x, v \in X^+, L[v] = 1, \{v, y\} \notin E\right\}| \tag{5}$$

$$\delta^+(y) = |\left\{v : v \in C_y, v \in X^-, L[v] = 0\right\}| \tag{6}$$

where C_x and C_y denote the set of neighboring vertices of x and y, respectively.

$\delta^-(x)$ counts the number of neighboring vertices of x which satisfy the following conditions:

- the vertex is in X^+.
- the vertex has $L[i] = 1$.
- the vertex does not connect to y.

A dominated vertex v will not be dominated after moving its unique neighboring vertex x in X. In addition, if the dominated vertex v in X^+ is adjacent to the swapped vertex y, it will be still dominated after performing the move. Thus, this kind of vertex should be excluded.

Conversely, $\delta^+(y)$ counts the number of neighboring vertices of y which satisfy the following conditions:

- the vertex is in X^-.
- the vertex has $L[i] = 0$.

A nondominated vertex v in X^- will be dominated by moving its neighboring vertex y into X. Thus, the time complexity of evaluating $\delta(m)$ is $O(|C_x| + |C_y|)$.

We now describe how list L is updated once a move is performed during the local search procedure. If a move $< x, y >$ is performed, only the elements corresponding to the neighbors of x and y need to be updated. Therefore, there are at most $|C_x| + |C_y|$ elements that need to be updated. Specifically, if move $< x, y >$ is applied, L can be updated as follows:

$$L[u] \leftarrow L[u] - 1 \quad \forall u \in C_x \tag{7}$$

$$L[v] \leftarrow L[v] + 1 \quad \forall v \in C_y \tag{8}$$

4. THE RSN-BASED TABU SEARCH ALGORITHM

In this section, we detail the proposed RSN-TS algorithm to solve the MCDSP. The most important procedure of this algorithm is the tabu search procedure, named TS_KCDS that exploits local search and tabu in order to tackle the k-CDSP.

Tabu Search (TS) was first proposed by [15] and, since then, it has been applied for successfully solving numerous combinatorial optimization problems (e.g., [20, 21, 26]). A tabu algorithm typically incorporates a tabu list as a recency-based memory structure to guarantee that solutions visited within a certain span of iterations will not be revisited. The algorithm then restricts consideration to moves not forbidden by the tabu list. Additional techniques frequently used in tabu search are diversification and intensification. In particular, the goal of diversification is to escape from the region currently visited by the algorithm, in order to explore new regions in the search space. In the following, we first describe the main procedure of the RSN-TS algorithm. Then, we describe the TS_kCDS procedure. In particular, we detail the particular diversification technique, named adaptive perturbation mechanism, used in this procedure.

4.1. The Main Framework

The RSN-TS algorithm tackles the MCDSP, that is, the problem to find a CDS of minimum cardinality in a given graph. The pseudocode of the RSN-TS algorithm is given in Algorithm 1.

Algorithm 1 Algorithm for the MCDS problem

```
1: procedure RNS_TS()
2:     X_best ← V
3:     iter ← 0
4:     while iter < nbIter do
5:         X ← REMOVE_VERTEX(X_best)
6:         (succ, X, iterTS) ← TS_KCDS(X, nbIter − iter)
7:         iter ← iter + iterTS
8:         if succ = true then
9:             X_best ← X
10:        end if
11:    end while
12:    return X_best
13: end procedure
```

In Algorithm 1, X_{best} records the best solution, that is, the smallest CDS found so far by the algorithm. At the beginning, X_{best} is initialized as the whole set of vertices (V). On each iteration, a randomly selected nonarticulation point is removed from X_{best}. The obtained connected set X of vertices is then transmitted to the TS_KCDS procedure whose role is to try to find a CDS of size $k = |X_{best}| - 1$. The smallest CDS found by the algorithm is returned at the end of the *nbIter* local search iterations, where *nbIter* is a parameter of the algorithm.

In the above description, the stopping criterion is the total number of iterations (*nbIter*), but it is easy to adapt the algorithm to use an alternative stopping criterion, such as the total computing time.

4.2. The Tabu Search Procedure

The TS_KCDS procedure has two input parameters: the initial configuration X_0 and the number of iterations *nbIter*. We denote by k the cardinality of $|X_0|$.

The goal of the procedure is to explore the set of configurations by applying a series of moves to discover a configuration of minimum cost. As already mentioned in Section 3, a configuration X corresponds to any connected set of vertices of size k, the cost of configuration X is the number of vertices of the graph that are not dominated by X, and the set of moves is defined by the restricted swap neighborhood (RSN).

In order to escape from local optimum, the TS_KCDS procedure employs a tabu mechanism based on a simple idea. This idea is to forbid a vertex recently removed from the configuration to be reinserted into the configuration for a short period. A vertex that belongs to the tabu list is said to be tabu. Also, any move $< x, y >$ such that y belongs to the tabu list is declared tabu. After a move $m =< x_m, y_m >$ is performed, vertex x_m is inserted into the tabu list for tt iterations, where tt (the so-called tabu tenure) is determined by two parameters named *ttMin* and *ttMax*.

In addition, the tabu status of a move $< x, y >$ will be overridden (by the so-called aspiration criterion) if applying this move leads to a configuration whose cost is smaller than the cost *fBest* of the best configuration. Note that this is the case if the following property holds: $f(X) + \delta(x, y) < fBest$. Finally, we denote any move that is non-tabu or that satisfies the aspiration criterion as a candidate move. On each iteration, the tabu algorithm will select the best candidate move, that is, the one having the lowest score.

The TS_KCDS procedure also employs a diversification technique that will be detailed below in Section 4.3. The pseudocode of the TS_KCDS procedure is given in Algorithm 2.

In Algorithm 2, X represents the current configuration of the tabu search procedure. X is first initialized with the initial configuration X_0, which is a parameter of the procedure. On each iteration, a candidate move $< x, y >$ is first selected. The selected move is the best candidate move applicable to X, that is, one that has the lowest score (ties are broken randomly).

Algorithm 2 Tabu search procedure for the k-CDS problem

```
1: procedure TS_KCDS(X_0, nbIter)
2:     INIT(X_0)
3:     for iter = 1 ... nbIter do
4:         < x, y > ← SELECT_MOVE()
5:         APPLY_MOVE(x, y)
6:         tt ← RAND(ttMin, ttMax)
7:         RENDER_MOVE_TABU(< x, y >, tt)
8:         if f(X) = 0 then
9:             return (true, X, iter)
10:        end if
11:        X ← APPLY_DIVERSIFICATION(X)
12:    end for
13:    return (false, X, iter)
14: end procedure
```

Then, move $< x, y >$ is applied to configuration X, that is, x is removed from X while y is added into X. Finally, vertex x is made tabu for tt iterations, such that tt is an integer randomly chosen in the range [*ttMin*, *ttMax*], where *ttMin* and *ttMax* are two parameters of the algorithm. The procedure stops when a zero-cost configuration is discovered or at the end of *nbIter* iterations.

In the next section, we describe the two procedures named INIT and APPLY_DIVERSIFICATION that implement the diversification technique.

4.3. The Diversification Technique

We employ in our tabu procedure a diversification technique, called the *adaptive perturbation mechanism*. This technique is inspired by the one exploited in the breakout local search algorithm [2–4]. It consists of applying periodically to the current configuration a perturbation whose strength depends on the search history. The idea is to perform in general a perturbation of moderate strength, except when the algorithm cycles, in which case the strength of the perturbation will be larger.

In our tabu procedure, a perturbation consists of a series of random moves and the strength of the perturbation is measured by the number of random moves to be performed. Therefore, the perturbation can be seen as a jump in the search space, and the strength of the perturbation as the amplitude of the jump.

The perturbation mechanism is governed by three parameters: *minStrength*, *maxStrength*, and *perturbPeriod*. Parameters *minStrength* and *maxStrength* are used to set the value of the perturbation strength, while parameter *perturbPeriod* is used to determine when a perturbation is triggered.

Also, the algorithm manipulates three symbols named X_{best}, *strength*, and *iterStagnation*. X_{best} denotes the best configuration; more precisely, it is the last best configuration visited by the procedure. Therefore, X_{best} is updated whenever $f(X) \leq f(X_{best})$. *strength* represents the current value of the perturbation strength. The value of *strength* is initialized to *minStr* at the beginning of the search. Then, it is reinitialized whenever the best configuration is improved. It is also reinitialized whenever X and X_{best} have the same score but do not coincide. Finally, it is incremented if X and X_{best}

coincide: note that this case indicates that the algorithm is cycling. *iterStagnation* represents the number of iterations elapsed since the last improvement of the best configuration or since the last perturbation was triggered.

In the tabu algorithm, a perturbation is triggered when *perturbPeriod* iterations have been performed without improving the best configuration. In this case, a perturbation of strength *strength* is applied to X_{best}.

The pseudocode of the procedures involved in the perturbation mechanism is given in Algorithms 3 and 4. Please note that symbols X, X_{best}, *strength*, *iterStagnation* as well as the parameters of the RSN-TS algorithm (mentioned in Section 4.4) represent global variables, which means that they can be directly used in the TS_κCDS procedure.

Algorithm 3 Procedure used to initialize the perturbation mechanism

```
1: procedure INIT(X_0)
2:     X_best ← X ← X_0
3:     strength ← minStrength
4:     iterStagnation ← 0
5: end procedure
```

Algorithm 4 Procedure used to apply the perturbation mechanism on each iteration

```
1: procedure APPLY_DIVERSIFICATION()
2:     if f(X) < f(X_best) then
3:         X_best ← X
4:         strength ← minStrength
5:     else if f(X) = f_best then
6:         iterStagnation ← iterStagnation + 1
7:         if X = X_best then
8:             strength ← min(strength + 1, maxStrength)
9:         else
10:            strength ← minStrength
11:            X_best ← X
12:        end if
13:    end if
14:    if iterStagnation ≤ perturbPeriod then
15:        X ← PERTURBATION(X_best, strength)
16:        iterStagnation ← 0
17:    end if
18: end procedure
```

4.4. Parameters of the RSN-TS Algorithm

In addition to the input graph, the input of the RSN-TS algorithm consists of the following parameters:

- the parameters *ttMin* and *ttMax* used to set the tabu list;
- the parameters *minStrength* and *maxStrength* used to set the value of perturbation strength;
- the parameter *perturbPeriod* used to determine when a perturbation is triggered;
- the parameter *nbIter* used for the stopping criterion.

Parameter *nbIter* depends on the available computing time. Tests performed in order to set the other parameters will be presented in Section 5.3.

5. COMPUTATIONAL RESULTS AND COMPARISONS

In this section, we present experiments performed in order to evaluate the performance of the RSN-TS algorithm. In these experiments, we use four datasets containing both random and real-world problem instances of various sizes and densities. The results obtained by the tabu algorithm are carefully analyzed and are compared with those of different algorithms and heuristics proposed in the literature.

In the following, we first describe the benchmarks and the algorithms used for comparison. Then, we present preliminary tests aimed at setting the parameters of the tabu algorithm. Finally, we present the results obtained by our algorithm on each of the four different datasets.

The RSN-TS algorithm is programmed in the Java language. All experiments have been executed on a PC with Intel i3 3.0GHz CPU and 4 GB RAM on Windows 8.1 with Oracle JRE 1.8.[1]

5.1. Benchmarks

For our tests, we have selected the MCDSP instances that are publicly available and for which results are presented in the literature. Our benchmarks include 97 problem instances coming from the four following datasets:

- **LPNMR graphs**: This dataset contains 9 problem instances. These problem instances have been proposed for a contest held in the context of the 10th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'09).
- **LMS graphs**: This dataset contains 41 problem instances whose optima are known. These problem instances have been proposed in [22]; we will name this dataset from the initials of the three authors (LMS).
- **BPFTC graphs**: This dataset contains 6 problem instances whose optima are known. These graphs belong to the Bus Power Flow Test Case (BPFTC) which is a subset of the Power Systems Test Case.[2] Each graph represents a portion of the American Electric Power System.
- **RGG graphs**: This dataset contains 41 large random geometric graphs (RGG) whose optima are unknown. A random geometric graph is obtained by randomly placing n vertices in a $N \times N$ square and connecting two vertices with an edge if their distance is smaller than a given threshold R (radius). These instances have been proposed in [18].

Details about these instances are given in Table 1. In the table, the different columns indicate for each dataset: the name of this dataset, the number of graphs it contains, the range of the values of the number of vertices, the range of the size of the best-known solutions, and the type of graph.

[1] The executable code and the detailed results obtained by our algorithm are available on request to the authors.

[2] http://www.ee.washington.edu/research/pstca/

TABLE 1. Description of the benchmarks

Dataset	#instances	$\|V\|$	Solution size	Type
LPNMR	9	[40,90]	[5,13]	Random
RGG	41	[400,3000]	[6,77]	Geometric
LMS	41	[30,200]	[2,27]	Random
BPFTC	6	[14,300]	[5,129]	Real-world

5.2. *Algorithms Used for Comparison*

In our experimental study, our RSN-TS algorithm will be compared with the following algorithms.

- **Exact algorithms**: Six state-of-the-art exact algorithms have been developed in [14]. These six algorithms are the stand-alone and iterative probing branch-and cut algorithms (SABC and IPBC), the stand-alone and iterative probing Benders decomposition algorithm (SABE and IPBE), and the stand-alone and iterative probing hybrid algorithms (SAHY and IPHY). Note that the SABC algorithm is derived from an algorithm proposed in [27]. These algorithms are implemented in C on a 2.0 GHz Intel XEON E5405 machine with 8 GB RAM [14].
- **Heuristics**: Two ant colony optimization (ACO) algorithms dedicated to the MCDSP have been recently developed and tested in [18]. These two heuristics are named the ACO Min-Max Ant System (ACO-MMAS) and the ACO with Pheromone Correction Strategy (ACO-PCS). According to [18], the ACO-PCS algorithm is very competitive on the LPNMR instances when compared to the algorithms that participated in the contest organized for the LPNMR'09 conference. The two ACO algorithms have been programmed in C# and run on a Dell OptiPlex 755 (Intel Core CPU E8500 3.16 GHz, 3 GB RAM) [18].

5.3. *Calibration of the Parameters of the Tabu Search Algorithm*

In this section, we present preliminary experiments conducted in order to fix the values of the key parameters of the RSN-TS algorithm.

- Parameters *ttMin* and *ttMax* determine the range of the tabu tenure (*tt*). We have tested three possible values for these parameters: $[ttMin, ttMax] = [1, 10]$, $[10, 50]$, and $[50, 100]$.
- Parameter *perturbPeriod* (line 14 in Algorithm 4) corresponds to the period used to apply a perturbation. We have tested three values for this parameter: $\alpha = 100, 300$, and 600.
- Parameters *minStrength* and *maxStrength* are used for the setting of the strength of the perturbation. We have tested two possible values for these parameters: $(minStrength, maxStrength) = (1, 10)$, and $(|V|/3, |V|)$.

For this experiment, we have used the 6 LMS instances with $|V| = 200$ as the test bed. These instances are random graphs whose density ranges from 5% to 70%. The algorithm was run for 10 times on each problem instance and each parameter setting. The time limit was set to 30 s for each run. The results of this experiment are presented in Table 2. Each row in the table corresponds to a particular combination of the parameters. The first columns indicate the combination of the parameters. The last two columns display, for the considered combination, the gap between the optimum and the average score of the solutions returned by the algorithm over the 10 runs, and the average running time to reach the best solution.

TABLE 2. Calibration experiment for parameter settings

Parameters			Average	
tt	*perturbPeriod*	*Strength*	Gap (%)	Time (s)
[1, 10]	100	(1, 10)	0.64	3.8
[1, 10]	100	$(\|V\|/3, \|V\|)$	0.64	4.8
[1, 10]	300	(1, 10)	0.64	6.7
[1, 10]	300	$(\|V\|/3, \|V\|)$	0.64	3.7
[1, 10]	600	(1, 10)	0.64	5.5
[1, 10]	600	$(\|V\|/3, \|V\|)$	0.64	4.5
[10, 50]	100	(1, 10)	0.64	2.8
[10, 50]	100	$(\|V\|/3, \|V\|)$	0.00	2.6
[10, 50]	300	(1, 10)	0.64	5.8
[10, 50]	300	$(\|V\|/3, \|V\|)$	0.00	3.3
[10, 50]	600	(1, 10)	0.64	3.8
[10, 50]	600	$(\|V\|/3, \|V\|)$	0.00	2.8
[50, 100]	100	(1, 10)	1.28	4.3
[50, 100]	100	$(\|V\|/3, \|V\|)$	1.28	2.7
[50, 100]	300	(1, 10)	1.28	3.7
[50, 100]	300	$(\|V\|/3, \|V\|)$	0.64	7.8
[50, 100]	600	(1, 10)	1.28	5.0
[50, 100]	600	$(\|V\|/3, \|V\|)$	1.28	5.0

From Table 2, we notice that there are only small differences in the performance of the algorithm between the different settings, which indicates the stability of the RSN-TS algorithm. However, we observe that the parameter setting ($[10, 50]$, 100, $(|V|/3, |V|)$) leads to the best performance.

5.4. *Experiments Performed with the RSN-TS Algorithm*

In our experiments, we have applied our RSN-TS algorithm to the 97 problem instances contained in our four datasets. The parameters of the algorithm have been fixed according to the best values found during the preliminary experiments reported in Section 5.3: $ttMin = 10, ttMax = 50, periodPerturbation = 100, strMin = |V|/3$, and $strMax = |V|$. In these experiments, except for LPNMR instances, the RSN-TS algorithm was run 20 times on each problem instance and the time limit for each run was set to 500 s. For the LPNMR set, the time limit was also set to 500 s per run; however, for a fair comparison with the reference algorithm, the RSN-TS algorithm was run only 10 times on each problem instance.

5.5. *Computational Results on LPNMR Instances*

Table 3 displays the results of the experiments performed with our RSN-TS algorithm on the LPNMR instances. The results obtained by the algorithm are compared to those of the two ACO algorithms (ACO-PCS and ACO-MMAS) presented in Section 5.2. In the table, the successive columns indicate for each instance: the name of the instance, the best results reported in the LPNMR'09 conference, and the results

TABLE 3. Computational results obtained on LPNMR instances

		RSN-TS				
Instance	LPNMR	Avg	σ	Iter	ACO MMAS	ACO PCS
40_200	5	5.0(1)	0	7	5.8(3)	5.3(4)
45_250	5	5.0(1)	0	24	5.8(3)	5.5(4)
50_250a	8	7.0[1] (1)	0	11	8.1(4)	8.0(6)
50_250b	7	7.0(1)	0	10	7.5(5)	7.1(6)
55_250	8	8.0(2)	0	37	8.8(5)	8.3(7)
60_400	7	7.0(2)	0	11	7.0(6)	7.0(9)
70_250	13	12.0[1] (4)	0	314	14.2(11)	13.9(13)
80_500	9	9.0(2)	0	17	10.0(12)	9.8(16)
90_600	10	9.0[1] (2)	0	525	10.9(14)	10.6(17)

[1]Updates the upper bound.

obtained for this instance by our RSN-TS, along with those obtained by ACO-PCS and ACO-MMAS. For the two ACO algorithms, the table displays the average size of the solutions returned by the algorithm and, in brackets, computational times in seconds for 10 runs. The results for ACO-MMAS and ACO-PCS are reproduced from [18]. For our RSN-TS algorithm, the table also reports the standard deviation of the results over 10 runs (denoted as σ) and, in brackets, computational times in seconds for 10 runs and the average number of local search iterations to reach the best results (denoted as *iter*).

From Table 3, we can observe that RSN-TS obtains the known best result on each run. We can also observe that our RSN-TS algorithm outperforms the two ACO algorithms with respect to solution quality.

5.6. Computational Results Obtained on LMS Instances

Table 4 displays the results of the experiments performed with our RSN-TS algorithm on the LMS instances. The results obtained by our algorithm are compared to those obtained by the six exact algorithms presented in Section 5.2. The results of the six exact algorithms are reproduced from [14]. Note that each exact algorithm was run (once) on each problem instance with a time limit set to 3,600 s. The value is indicated in bold if it hits the optimum. This applies to all the following tables.

In Table 4, each line corresponds to a particular instance and the successive columns indicate the name of this instance, the optimal value of the solution, the results of our RSN-TS algorithm and those of the six exact algorithms. For RSN-TS, the table displays the average score of the solutions obtained by the algorithm (over the 20 runs), the average computing time, the standard deviation, and the average number of iterations. For the six exact algorithms, the table displays the score of the best solution discovered by the algorithm and the total time used to prove the optimality of the solution. The designation "*" indicates that the algorithm was not able to terminate before the time limit. Note however that, in this case, it may have discovered an optimal solution within the time limit but was not able to prove the optimality of the found solution. Also note that, when the algorithm finished before the time limit, the displayed time corresponds to the time needed to prove optimality, while the time to find an optimal solution is generally smaller.

From Table 4, we can observe that for the same instance, the results obtained by the exact algorithms can be very different. Let us consider, for example, the instance named "v70_d5" whose optimum equals 27. For this small instance of only 70 vertices, three of the six exact algorithms did not find a solution of size 27, while two other ones found an optimal solution within only one second or less. The number of instances for which a given algorithm has not reached the optimum value during the time limit (whether optimality was proven or not) equals 3, 4, 2, 1, 3, and 1 for SABC, IPBC, SABE, IPBE, SAHY, and IPHY, respectively. Therefore, we notice that none of the 6 exact algorithms could reach the optimal solutions for all the problem instances within the time limit. Conversely, we can notice that, for each problem instance, at least one exact algorithm was able to find an optimal solution. In this respect, the most difficult problem instance is v200_d5. For this problem instance, only one exact algorithm could reach the optimum (IPBE) and the computing time needed to prove the optimality of the solution amounts to 1,658 s. If we turn to the results obtained by the RSN-TS algorithm, we can notice that the minimum value of the solutions produced by the tabu algorithm equals the optimum for each problem instance. Also, we can observe that the average computing time to reach the best solution is at most equal to 4 s. These results indicate that while these problem instances look difficult for exact algorithms, they are solved very efficiently and very quickly by the proposed tabu algorithm.

5.7. Computational Results Obtained on BPFTC Instances

Table 5 displays the results of the experiments performed with our RSN-TS algorithm on the BPFTC instances. The results obtained by the algorithm are compared to those of the six exact algorithms presented in Section 5.2. The results of the six exact algorithms are reproduced from [14]. Each exact algorithm was run once on each problem instance with a time limit set to 3,600 s.

From the table, as far as the first five problem instances are concerned, we can make similar remarks as those we made previously for the LMS problem instances. The last instance (ieee_300_bus) looks much more difficult. The best-performing exact algorithm is IPBC and the score of the best solution found by this algorithm is 136 while the optimum equals 129. Conversely, our RSN-TS algorithm was able to find an optimal solution on each run.

5.8. Computational Results Obtained on RGG Instances

Table 6 displays the results of the experiments performed with our RSN-TS algorithm on the RGG problem instances. Note that these instances are much larger than those involved in the preceding experiments, as these graphs have between

TABLE 4. Computational results obtained on LMS instances

		RSN-TS								
Instance	OPT	avg	σ	iter	SAHY	IPHY	SABE	IPBE	SABC	IPBC
v30_d10	15	**15.0(<1)**	0	24	15(12)	15(8)	15(41)	15(24)	15(<1)	15(<1)
v30_d20	7	**7.0(<1)**	0	11	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)
v30_d30	4	**4.0(<1)**	0	64	4(<1)	4(<1)	4(<1)	4(<1)	4(<1)	4(<1)
v30_d50	3	**3.0(<1)**	0	3	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v30_d70	2	**2.0(<1)**	0	3	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v50_d5	31	**31.0(<1)**	0	33	31(22)	31(9)	31(*)	31(*)	31(<1)	31(<1)
v50_d10	12	**12.0(<1)**	0	211	12(2)	12(1)	12(12)	12(1)	12(<1)	12(<1)
v50_d20	7	**7.0(<1)**	0	10	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)
v50_d30	5	**5.0(<1)**	0	8	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)
v50_d50	3	**3.0(<1)**	0	6	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v50_d70	2	**2.0(<1)**	0	3	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v70_d5	27	**27.0(<1)**	0	741	29(*)	27(290)	29(*)	29(*)	27(1)	27(<1)
v70_d10	13	**13.0(<1)**	0	101	13(25)	13(1)	13(1)	13(1)	13(14)	13(5)
v70_d20	7	**7.0(<1)**	0	274	7(1)	7(<1)	7(<1)	7(<1)	7(2)	7(1)
v70_d30	5	**5.0(<1)**	0	52	5(<1)	5(<1)	5(<1)	5(<1)	5(1)	5(<1)
v70_d50	3	**3.0(<1)**	0	6	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v70_d70	2	**2.0(<1)**	0	82	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v100_d5	24	**24.0(1)**	0	574	24(970)	24(35)	24(2542)	24(1963)	24(38)	24(65)
v100_d10	13	**13.0(1)**	0	110	13(2)	13(1)	13(<1)	13(<1)	13(17)	13(35)
v100_d20	8	**8.0(1)**	0	72	8(6)	8(2)	8(1)	8(1)	8(205)	8(534)
v100_d30	6	**6.0(1)**	0	21	6(11)	6(4)	6(3)	6(2)	6(209)	6(275)
v100_d50	4	**4.0(1)**	0	6	4(3)	4(1)	4(1)	4(<1)	4(40)	4(27)
v100_d70	3	**3.0(1)**	0	2	3(1)	3(<1)	3(1)	3(<1)	3(12)	3(12)
v120_d5	25	**25.0(1)**	0	363	25(1118)	25(27)	25(3)	25(10)	25(705)	25(105)
v120_d10	13	**13.0(1)**	0	1,416	13(56)	13(18)	13(3)	13(3)	13(*)	15(*)
v120_d20	8	**8.0(2)**	0	241	8(16)	8(8)	8(5)	8(3)	8(828)	8(*)
v120_d30	6	**6.0(1)**	0	141	6(14)	6(7)	6(5)	6(4)	6(496)	6(1039)
v120_d50	4	**4.0(1)**	0	4	4(8)	4(4)	4(4)	4(2)	4(161)	4(153)
v120_d70	3	**3.0(1)**	0	2	3(2)	3(2)	3(2)	3(<1)	3(34)	3(26)
v150_d5	26	**26.0(2)**	0	703	27(*)	26(*)	26(1047)	26(259)	27(*)	27(*)
v150_d10	14	**14.0(4)**	0	4,993	14(651)	14(195)	14(51)	14(28)	15(*)	15(*)
v150_d20	9	**9.0(2)**	0	65	9(2117)	9(902)	9(366)	9(273)	9(*)	9(*)
v150_d30	6	**6.0(3)**	0	111	6(34)	6(24)	6(21)	6(11)	6(2077)	6(*)
v150_d50	4	**4.0(1)**	0	29	4(17)	4(10)	4(7)	4(5)	4(535)	4(342)
v150_d70	3	**3.0(3)**	0	3	3(5)	3(2)	3(4)	3(<1)	3(51)	3(45)
v200_d5	27	**27.0(4)**	0	8,705	29(*)	28(*)	29(*)	27(1658)	29(*)	29(*)
v200_d10	16	**16.0(2)**	0	468	16(*)	16(*)	16(*)	16(*)	16(*)	16(*)
v200_d20	9	**9.0(2)**	0	289	9(*)	9(*)	9(1687)	9(1943)	9(*)	9(*)
v200_d30	7	**7.0(2)**	0	22	7(*)	7(*)	7(3208)	7(1848)	7(*)	7(*)
v200_d50	4	**4.0(2)**	0	64	4(41)	4(28)	4(24)	4(19)	4(2247)	4(1279)
v200_d70	3	**3.0(2)**	0	3	3(9)	3(5)	3(10)	3(<1)	3(334)	3(261)

TABLE 5. Computational results obtained on BPFTC instances

		RSN-TS								
Instance	OPT	Avg	σ	Iter	SAHY	IPHY	SABE	IPBE	SABC	IPBC
ieee_14_bus	5	**5.0(<1)**	0	6	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)
ieee_30_bus	11	**11.0(<1)**	0	80	11(<1)	11(<1)	11(<1)	11(<1)	11(<1)	11(<1)
ieee_57_bus	31	**31.0(<1)**	0	39	31(24)	31(192)	31(*)	31(*)	31(<1)	31(<1)
RTS96	32	**32.0(1)**	0	2,545	32(118)	32(26)	34(*)	34(*)	32(2)	32(4)
ieee_118_bus	43	**43.0(1)**	0	106	43(65)	43(2)	43(*)	43(*)	43(<1)	43(<1)
ieee_300_bus	129	**129.0**[a] **(57)**	0	38,185	139(*)	138(*)	139(*)	139(*)	139(*)	136(*)

[a]Updates the upper bound.

400 and 3,000 vertices—versus a maximum of 200 vertices in the graphs considered so far, except one graph having 300 vertices (ieee_300_bus). As the optima of these instances are unknown, we compare the results of the tabu algorithm to a lower bound calculated using the method proposed in [22]. The results obtained by the RSN-TS algorithm are

TABLE 6. Computational results obtained on RGG instances

		RSN-TS					ACO-MMAS		ACO-PCS	
Instance	LB	Best	Avg	σ	Iter	Time(s)	Best	Diff	Best	Diff
n400_80_r60	15	18[a]	18.0	0	417	<1	20	2	19	1
n400_80_r70	12	14[a]	14.0	0	843	<1	16	2	15	1
n400_80_r80	10	12	12.0	0	145	<1	12	0	12	0
n400_80_r90	8	10[a]	10.0	0	1,555	<1	11	1	11	1
n400_80_r100	7	8	8.0	0	1,195	<1	8	0	8	0
n400_80_r110	6	7[a]	7.0	0	7,180	2	8	1	8	1
n400_80_r120	6	**6**[a]	6.0	0	2,209	1	7	1	7	1
n600_100_r80	18	21[a]	21.0	0	903	<1	23	2	22	1
n600_100_r90	16	19[a]	19.0	0	2,011	1	22	3	21	2
n600_100_r100	14	16[a]	16.0	0	1,927	1	17	1	17	1
n600_100_r110	12	15	15.0	0	443	<1	15	0	15	0
n600_100_r120	10	13[a]	13.0	0	747	<1	15	2	14	1
n700_200_r70	31	39[a]	39.0	0	14,714	17	46	7	46	7
n700_200_r80	26	32[a]	32.0	0	12,525	15	41	9	41	9
n700_200_r90	21	26[a]	26.0	0	3,442	5	34	8	33	7
n700_200_r100	18	22[a]	22.0	0	45,261	61	28	6	28	6
n700_200_r110	16	20[a]	20.0	0	1,174	3	23	3	22	2
n700_200_r120	13	17[a]	17.0	0	7,808	11	21	4	21	4
n1000_200_r100	31	38[a]	38.4	0.63	104,418	124	46	8	46	8
n1000_200_r110	27	34[a]	34.0	0	7,531	9	43	9	42	8
n1000_200_r120	24	29[a]	29.0	0	75,985	86	37	8	37	8
n1000_200_r130	21	26[a]	26.0	0	9,370	12	32	6	32	6
n1000_200_r140	18	23[a]	23.0	0	3,435	5	30	7	29	6
n1000_200_r150	17	21[a]	21.0	0	6,483	9	28	7	26	5
n1000_200_r160	15	19[a]	19.0	0	55,655	71	24	5	25	6
n1500_250_r130	40	49[a]	49.0	0	62,371	111	60	11	60	11
n1500_250_r140	36	44[a]	44.0	0	10,223	19	53	9	52	8
n1500_250_r150	32	40[a]	40.0	0	47,039	91	51	11	51	11
n1500_250_r160	29	36[a]	36.0	0	16,886	31	47	11	45	9
n2000_300_r200	33	42[a]	42.0	0	51,191	136	55	13	52	10
n2000_300_r210	30	38[a]	38.3	0.55	36,387	101	51	13	50	12
n2000_300_r220	28	35[a]	35.0	0	48,124	134	47	12	45	10
n2000_300_r230	26	33[a]	33.0	0	92,929	284	44	11	44	11
n2500_350_r200	49	61[a]	61.1	0.32	33,142	120	79	18	79	18
n2500_350_r210	45	56[a]	56.0	0	32,232	119	75	19	74	18
n2500_350_r220	42	52[a]	52.0	0	33,145	123	68	16	69	17
n2500_350_r230	39	50[a]	50.0	0	35,385	133	66	16	66	16
n3000_400_r210	61	75[a]	75.3	0.55	57,374	257	99	24	98	23
n3000_400_r220	56	71[a]	71.0	0	32,334	129	88	17	91	20
n3000_400_r230	53	65[a]	65.6	0.77	32,334	149	86	21	86	21
n3000_400_r240	49	61[a]	61.5	0.71	55,634	258	82	21	80	19

[a]Updates the upper bound.

compared to those of the two ACO algorithms (ACO-PCS and ACO-MMAS) presented in Section 5.2. The results for ACO-MMAS and ACO-PCS are reproduced from [18]. In the table, the successive columns display: the name of the instance; the lower bound; for the RSN-TS algorithm, the score of the best solution obtained by the algorithm, the average score, the standard deviation of the scores (σ) as well as the average number of iterations and computational time consumed by RSN-TS; for ACO-MMAS and ACO-PCS, the best score reached by the ACO algorithm and the difference between this result and the minimum score obtained by RSN-TS. The dagger symbol means that the RSN-TS has improved the best solution found by ACOs.

From Table 6, we can notice that the gap between the minimum score reached by our algorithm and the lower bound generally ranges between 15% and 30%. As the values of the gap are relatively high, it does not tell us very much about the performance of the tabu algorithm. If we compare the difference between the minimum score of the tabu algorithm and the one of the two ACO algorithms, we can observe that it ranges between 0 and up to 24 and 23 for ACO-MMAS and ACO-PCS, respectively. The difference tends to increase unsurprisingly with the order of the graph. For graphs of same order, the difference tends to decrease with the density; in other words, as a smaller density tends to translate into a larger minimum CDS, the difference increases with the size of the solution. If we consider the largest and sparsest graph (n3000_400_r210), we notice that the best solutions produced by the two ACO algorithms are 29% and 27% larger than the one produced by RSN-TS (99 and 98, vs. 75). This

indicates that, for these large instances, the tabu search algorithm outperforms the two ACO algorithms by a very large margin. In addition, we can observe that the standard deviation (σ) equals zero for all the problem instances, except for a few ones where it is very small. In particular, it is much smaller than the difference between the average quality of the solutions obtained by our TS algorithm and the ones obtained by ACO. This indicates that the results are very stable and that the dominance of our algorithm over ACO is real, and not simply due to chance.

It is important to observe that the proposed RSN-TS algorithm is a stochastic algorithm, which means that it performs some random choices. More precisely, there are four different sources of randomness in the tabu algorithm: in the selection of a move (in procedure *select_move* () called in Algorithm 2, line 4); in the computing of the tabu tenure (Algorithm 2, line 6); when a vertex is removed from the solution (in procedure *remove_vertex* () called in Algorithm 1, line 5); and, finally, during diversification (in procedure *perturbation* () called in Algorithm 4, line 15). As a result of these different random choices, when the algorithm is applied several times to the same problem instance (while using the same parameters), it is able to produce different solutions, and it actually generally does. For the LMS and BPFTC datasets, we have observed in a previous section that the RSN-TS algorithm produced for each instance an optimal solution on each run. For RGG instances, the fact that the algorithm again often produces solutions of the same quality may indicate that it also hits the optimum, although we cannot be sure of that, as the optimum is not known for these instances.

6. DISCUSSION AND ANALYSIS

We now turn our attention to discussing and analyzing two important features of the proposed RSN-TS algorithm, namely the proposed incremental evaluation technique and the diversification techniques. Moreover, we discuss whether the algorithm can be improved by using a more sophisticated initialization phase.

6.1. *Importance of the Incremental Evaluation Technique*

The main ingredient of our RSN-TS algorithm is the RS-based neighborhood structure. For a neighborhood search method, it is particularly important to be able to rapidly evaluate the neighborhoods. As described in Section 3.3, we propose an incremental evaluation technique for evaluating the neighborhood moves, where a list L with length $|V|$ is maintained to calculate the move values of all possible candidate moves. Once a move is performed, only the values of list L affected by the move are accordingly updated.

In order to evaluate the effectiveness of this incremental evaluation technique, we carry out computational experiments to compare the performance of the RSN-TS algorithm with and without this technique on some representative instances, which are v100_d70, v200_d5, ieee_57_bus and ieee_300_bus. Note that similar results can be observed on other instances. The following two strategies are considered: Our fast incremental updating strategy used in this paper, called fast evaluation strategy (FES); the fast incremental evaluation technique is not used and all the neighborhood moves are calculated from scratch, called all calculating strategy (ACS). The evolution of the average CPU time with the number of iterations is shown in Figure 2.

One finds that the RSN-TS with FES is much faster than that with ACS. For example, for the instance ieee_300_bus, the FES strategy is about 20 times faster than the ACS strategy. Moreover, it should be stated that the larger the size of V, the more efficient the FES strategy is. This experiment clearly demonstrates the importance of the fast evaluation strategy proposed in this article.

6.2. *Influence of the Diversification Techniques*

The RSN-TS algorithm exploits two kinds of diversification techniques: a tabu list (for short term diversification) and a perturbation mechanism (for long-term diversification). In order to evaluate the influence of these two techniques and how they contribute to the quality of the results, we have performed a series of new experiments. In these experiments, we have tested, in addition to RSN-TS, three other variants denoted by RSN-TS $_{np}$, RSN-LS, and RSN-LS $_{np}$. RSN-LS is the algorithm obtained by replacing the tabu operator by pure local search (i.e., iterative improvement) while the extension "*np*" stands for "no perturbation," which indicates that the perturbation mechanism is not used. The experiment was carried out on 18 LMS and 20 RGG benchmarks as displayed in Tables 7 and 8, respectively, where the best, average results, and the standard deviations are reported for each variant. In addition, the last three rows present the number of better, equal, and worse results of RSN-TS in terms of all the three criteria compared with other three variants. In each dataset (LMS and RGG), the tables give the results obtained for the largest problem instances; however, we have observed the same tendencies for the other (smaller) problem instances.

Let us first observe in Table 7 the results obtained with the 18 LMS instances. For these instances, the original version of the algorithm (RSN-TS) reached the optimum on each run. These instances can thus be considered to be very easy for RSN-TS. RSN-LS also reached the optimum on each run, but only for 13 out of the 18 displayed instances; even worse results have been obtained by RSN-TS $_{np}$ and, to a larger extent, by RSN-LS $_{np}$. If we consider, in Table 8, the 20 RGG problem instances, we notice a clear dominance of RSN-TS over the three other variants, as RSN-TS obtained a lower minimum and a lower average score for most problem instances. We also notice that RSN-TS has a very low standard deviation for all problem instances, and even a standard deviation equal to 0 for 14 of them (out of 20). This may indicate that, at least for these 14 problem instances, RSN-TS was very robust in obtaining high quality solutions. By contrast, the other variants display in general much larger standard deviations. In summary, the above-reported

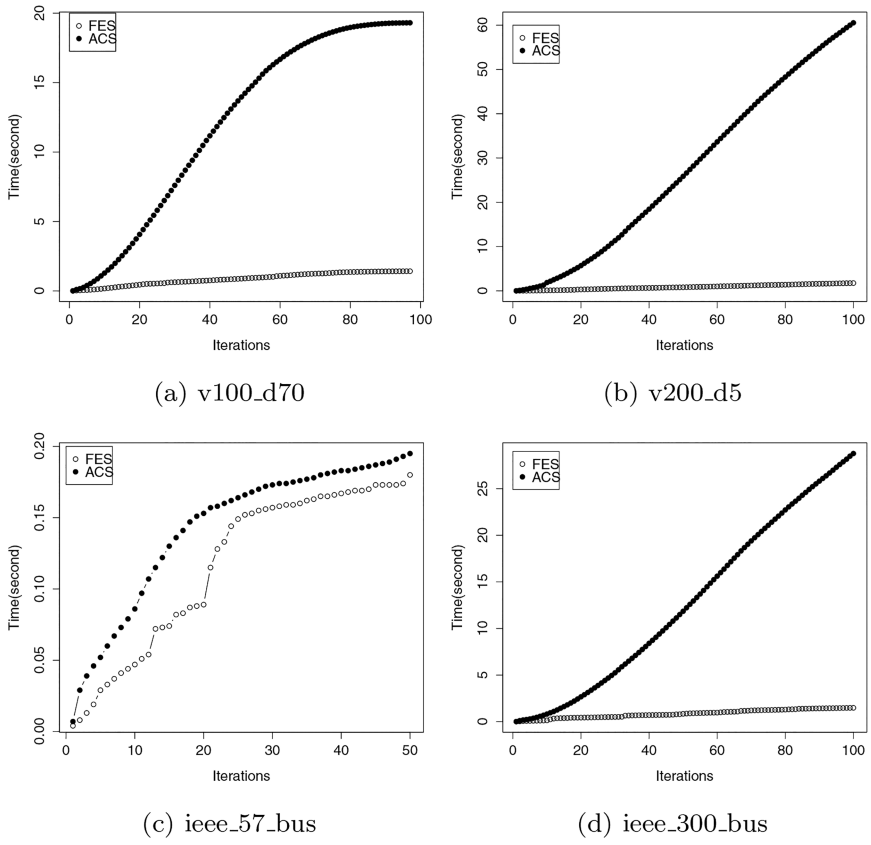

FIG. 2. Computational performance comparison with and without the fast neighborhood evaluation strategy.

TABLE 7. Comparison between RSN-TS, RSN-TS $_{np}$, RSN-LS and RSN-LS $_{np}$: LMS problem instances

	RSN-TS			RSN-TS $_{np}$			RSN-LS			RSN-LS $_{np}$		
Instance	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ
v120_d5	25	25	0	25	25.9	1.10	25	25.3	0.48	26	31.8	3.05
v120_d10	13	13	0	13	13.7	0.95	13	13	0	15	16.8	1.14
v120_d20	8	8	0	8	8.6	0.52	8	8	0	8	9.8	1.03
v120_d30	6	6	0	6	6.2	0.42	6	6	0	6	7.1	0.57
v120_d50	4	4	0	4	4	0	4	4	0	4	4.1	0.32
v120_d70	3	3	0	3	3	0	3	3	0	3	3	0
v150_d5	26	26	0	26	26.6	0.70	26	26.1	0.32	30	33.1	2.42
v150_d10	14	14	0	15	15.8	0.92	14	14.9	0.32	18	19	0.94
v150_d20	9	9	0	9	9.3	0.48	9	9	0	10	11.3	0.82
v150_d30	6	6	0	6	6.5	0.53	6	6	0	7	8	0.67
v150_d50	4	4	0	4	4	0	4	4	0	4	4.5	0.53
v150_d70	3	3	0	3	3	0	3	3	0	3	3	0
v200_d5	27	27	0	27	29.9	1.60	28	28.2	0.42	35	38	2.54
v200_d10	16	16	0	16	16.9	0.88	16	16.1	0.32	20	21.1	0.74
v200_d20	9	9	0	9	9.6	0.70	9	9	0	10	11.3	0.67
v200_d30	7	7	0	7	7.3	0.67	7	7	0	7	7.7	0.48
v200_d50	4	4	0	4	4.1	0.32	4	4	0	4	4.9	0.32
v200_d70	3	3	0	3	3	0	3	3	0	3	3.1	0.32
better	—	—	—	1	5	—	1	5	—	9	17	—
equal	—	—	—	17	13	—	17	13	—	9	1	—
worse	—	—	—	0	0	—	0	0	0	0	0	—

TABLE 8. Comparison between RSN-TS, RSN-TS $_{np}$, RSN-LS and RSN-LS $_{np}$: RGG problem instances

	RSN-TS			RSN-TS $_{np}$			RSN-LS			RSN-LS $_{np}$		
Instance	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ
n1000_200_r100	38	38.4	0.63	40	43.3	2.79	38	40.6	1.35	46	52.3	2.58
n1000_200_r110	34	34	0	34	37.9	1.60	34	35.3	0.67	44	46.2	2.04
n1000_200_r120	29	29	0	31	34.6	1.96	30	30.8	0.79	37	41.1	3.41
n1000_200_r160	19	19	0	20	22	1.56	19	19.9	0.32	25	27.4	1.65
n1500_250_r130	49	49	0	52	57.3	3.71	52	53	1.05	61	67.7	3.74
n1500_250_r140	44	44	0	47	48.8	1.55	45	46.9	0.74	56	58.8	2.25
n1500_250_r150	40	40	0	44	46.1	1.73	42	43.4	1.17	52	55.1	1.91
n1500_250_r160	36	36	0	39	41	1.41	37	38.2	1.32	49	52.3	2.79
n2000_300_r200	42	42	0	43	48.4	3.13	45	45.6	1.07	56	59.3	2.06
n2000_300_r210	38	38.3	0.63	40	43.2	1.75	40	41.2	0.92	52	55.6	2.76
n2000_300_r220	35	35	0	40	41	1.33	36	37.2	1.03	45	50.5	2.95
n2000_300_r230	33	33	0	37	39.2	1.93	34	35.4	0.97	44	49.1	3.38
n2500_350_r200	61	61.1	0.32	66	68	2.16	64	66.8	1.93	80	84.1	3.11
n2500_350_r210	56	56	0	60	62.9	2.28	60	61.4	1.17	74	78.1	3.00
n2500_350_r220	52	52	0	57	60.3	1.89	56	58.4	1.26	70	73.3	2.95
n2500_350_r230	50	50	0	52	57.8	2.57	51	55.3	1.89	69	72	2.11
n3000_400_r210	75	75.3	0.55	80	85.7	3.40	81	83.9	1.73	99	102.9	2.64
n3000_400_r220	71	71	0	74	82.3	4.55	76	77.8	1.62	90	97.8	4.13
n3000_400_r230	65	65.6	0.77	68	72.3	2.50	70	71.5	1.27	83	88.5	2.99
n3000_400_r240	61	61.5	0.71	65	70.8	3.39	66	67.8	1.14	81	88	4.45
better	—	—	—	19	20	—	17	20	—	20	20	—
equal	—	—	—	1	0	—	3	0	—	0	0	—
worse	—	—	—	0	0	—	0	0	—	0	0	—

experiments show that both the tabu list and the perturbation mechanism play a crucial role in the efficiency of our algorithm.

6.3. *Discussion of the Initialization Phase*

As mentioned earlier, our RSN-TS algorithm simply uses the whole vertex set V as the initial solution, since the whole vertex set will certainly be a CDS. However, for a metaheuristic algorithm a better initial solution usually yields better final results. Therefore, it is natural to ask whether if it would be better to use a fast greedy heuristic to obtain an initial CDS instead of simply using the whole vertex set.

To test this idea, we implement the following experiment. We first obtain a k value using a fast greedy algorithm introduced in [5]. Then, we compare the time consumed by this greedy algorithm and RSN-TS to get the solution with size of k. Finally, we estimate the efficiency improvement if the greedy algorithm is used as the initialization phase of RSN-TS. The experiment is implemented on several hard instances presented in Section 5.6. The results are shown in Table 9. The second column k represents the best results that the greedy algorithm can obtain. The third and fourth columns represent the time (measured in milliseconds) that the greedy algorithm and RSN-TS use to obtain the result k, respectively. The fifth column represents the time that RSN-TS uses to obtain the best results. The last column represents the percentage of efficiency improvement obtained by using the greedy algorithm as the initialization phase in RSN-TS.

From Table 9, one observes that using a greedy algorithm as the initialization phase of RSN-TS can improve the computational efficiency to some extent. The degree of the improvement depends on the instance density. On high density graphs the improvement is prominent, while it is less so on sparse graphs. Moreover, one finds that if the greedy initialization phase is used there is no improvement for the final results. Thus, the improvement brought by the greedy initialization phase is limited. This further indicates the effectiveness and efficiency of the proposed restricted swap-based neighborhood.

7. CONCLUSION

In this article, we have proposed a restricted swap-based neighborhood structure and embedded it into a tabu search procedure for tackling the MCDSP. The performance of the proposed algorithm is assessed on four sets of instances (97 in total) widely used in the literature. Computational results show that the proposed algorithm is highly competitive in comparison with the state-of-the-art algorithms in the literature. By comparing with 9 reference state-of-the-art algorithms, the RSN-TS cannot only obtain competitive results, but also needs very short computational time. In particular, the RSN-TS improves the upper bounds for 41 out of the 97 instances, while matching the best known results for all the remaining the instances.

Further analysis indicates that the incremental evaluation technique proposed in this article is essential to enhance the computational efficiency of the proposed algorithm. In addition, the adaptive perturbation phase is a key feature to influence the performance of the proposed algorithm. Finally, a more sophisticated initialization phase can speed up the

TABLE 9. Comparison between two initialization techniques

Instance	k	Greedy (ms)	RSN-TS to k (ms)	RSN-TS to best (ms)	Imp (%)
v150_d5	97	16	184	2,864	5.9
v150_d10	57	18	714	4,168	16.7
v150_d30	17	12	581	2,945	19.3
v200_d5	110	9	289	3,654	7.7
v200_d10	48	13	333	2,987	10.7
v200_d30	15	10	496	2,883	16.9
v200_d50	9	12	577	2,812	20.9
v200_d70	7	28	650	2,689	23.1
ieee_300_bus	296	13	96	57,645	0.1

algorithm to some extent, while there is not an obvious improvement for the final results.

There are several directions to extend this work. First of all, more challenging instances should be developed to test the scalability of the algorithm. The other direction is to develop more semantic neighborhood move operators and perturbation operators that consider more problem specific knowledge. In addition, it is interesting to test the proposed neighborhood structure in other meta-heuristic frameworks, such as simulated annealing, GRASP [32], or distributed algorithms. Finally, our proposed local search can also be effectively combined with other population-based meta-heuristic approaches, such as evolutionary algorithm, path relinking, and so on [21, 26].

REFERENCES

[1] B. Balasundaram and S. Butenko, "Graph domination, coloring and cliques in telecommunications," Handbook of optimization in telecommunications, Mauricio G. C. Resende and Panos M. Pardalos (Editors), Springer, New York, 2006, pp. 865–890.

[2] U. Benlic and J.K. Hao, Breakout local search for maximum clique problems, Comput Oper Res 40 (2013), 192–206.

[3] U. Benlic and J.K. Hao, Breakout local search for the max-cut problem, Eng Appl Artif Intell 26 (2013), 1162–1173.

[4] U. Benlic and J.K. Hao, Breakout local search for the quadratic assignment problem, Appl Math Comput 219 (2013), 4800–4815.

[5] S. Butenko, X. Cheng, C. Oliveira, and P. Pardalos, "A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks," Recent developments in cooperative control and optimization, S. Butenko, R. Murphey, and P. Pardalos (Editors), Springer, New York, 2004, pp. 61–73.

[6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks, Wireless Netw 8 (2002), 481–494.

[7] S. Chen, I. Ljubić, and S. Raghavan, The regenerator location problem, Networks 55 (2010), 205–220.

[8] X. Cheng, M. Ding, D.H. Du, and X. Jia, Virtual backbone construction in multihop ad hoc wireless networks, Wireless Commun Mobile Comput 6 (2006), 183–190.

[9] B. Deb, S. Bhatnagar, and B. Nath, Multi-resolution state retrieval in sensor networks, Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, Anchorage, AK, USA, 2003, pp. 19–29.

[10] M. Ding, X. Cheng, and G. Xue, Aggregation tree construction in sensor networks, IEEE 58th Vehicular Technology Conference, Orlando, FL, USA, 2003, pp. 2168–2172.

[11] T. Fujie, An exact algorithm for the maximum leaf spanning tree problem, Comput Oper Res 30 (2003), 1931–1944.

[12] T. Fujie, The maximum-leaf spanning tree problem: Formulations and facets, Networks 43 (2004), 212–223.

[13] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, WH Freeman, New York, 1979.

[14] B. Gendron, A. Lucena, A.S. da Cunha, and L. Simonetti, Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem, INFORMS J Comput 26 (2014), 645–657.

[15] F. Glover and M. Laguna, "Tabu search," Handbook of combinatorial optimization, D.Z. Du and P. Pardalos (Editors), Springer, New York, 1999, pp. 2093–2229.

[16] S. Guha and S. Khuller, Approximation algorithms for connected dominating sets, Algorithmica 20 (1998), 374–387.

[17] J. Hopcroft and R. Tarjan, Algorithm 447: Efficient algorithms for graph manipulation, Commun ACM 16 (1973), 372–378.

[18] R. Jovanovic and M. Tuba, Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem, Comput Sci Inform Syst 10 (2013), 133–149.

[19] S. Leu and R.S. Chang, A weight-value algorithm for finding connected dominating sets in a MANET, J Netw Comput Appl 35 (2012), 1615–1619.

[20] Z. Lü and J.K. Hao, Adaptive tabu search for course timetabling, Eur J Oper Res 200 (2010), 235–244.

[21] Z. Lü and J.K. Hao, A memetic algorithm for graph coloring, Eur J Oper Res 203 (2010), 241–250.

[22] A. Lucena, N. Maculan, and L. Simonetti, Reformulations and solution algorithms for the maximum leaf spanning tree problem, Comput Manage Sci 7 (2010), 289–311.

[23] M.V. Marathe, H. Breu, H.B. Hunt, S.S. Ravi, and D.J. Rosenkrantz, Simple heuristics for unit disk graphs, Networks 25 (1995), 59–68.

[24] R. Misra and C. Mandal, Minimum connected dominating set using a collaborative cover heuristic for ad hoc

sensor networks, IEEE Trans Parallel Distrib Syst 21 (2010), 292–302.

[25] M. Morgan and V. Grout, Metaheuristics for wireless network optimisation, Third Advanced International Conference Telecommunications, Morne, Mauritius, 2007, pp. 15–15.

[26] B. Peng, Z. Lü, and T. Cheng, A tabu search/path relinking algorithm to solve the job shop scheduling problem, Comput Oper Res 53 (2015), 154–164.

[27] L. Simonetti, A.S. Da Cunha, and A. Lucena, "The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm," Network optimization, LNCS 6701, Springer, New York, 2011, pp. 162–169.

[28] Y.C. Tseng, S.Y. Ni, Y.S. Chen, and J.P. Sheu, The broadcast storm problem in a mobile ad hoc network, Wireless Netw 8 (2002), 153–167.

[29] P.J. Wan, K.M. Alzoubi, and O. Frieder, Distributed construction of connected dominating set in wireless ad hoc networks, Mob Netw Appl 9 (2004), 141–149.

[30] F. Wang, M.T. Thai, and D.Z. Du, On the construction of 2-connected virtual backbone in wireless networks, IEEE Trans Wireless Commun 8 (2009), 1230–1237.

[31] J. Wu and F. Dai, Broadcasting in ad hoc networks based on self-pruning, Int J Foundations Comput Sci 14 (2003), 201–221.

[32] X. Wu, T. Ye, Q. Guo, and Z. Lü, GRASP for traffic grooming and routing with simple path constraints in WDM mesh networks, Comput Netw 86 (2015), 27–39.

[33] J. Yu, N. Wang, G. Wang, and D. Yu, Connected dominating sets in wireless ad-hoc and sensor networks: A comprehensive survey, Comput Commun 36 (2013), 121–134.