Vertex Weighting-Based Tabu Search for p-Center Problem

Qingyun Zhang[1] , **Zhipeng Lü**[1] , **Zhouxing Su**[1*] , **Chumin Li**[2] , **Yuan Fang**[1] and **Fuda Ma**[3]
[1]SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]MIS, Université de Picardie Jules Verne, France
[3]Huawei Cloud Alkaid Lab, Huawei Technologies Co., Ltd., China
suzhouxing@hust.edu.cn

Abstract

The p-center problem consists of choosing p centers from a set of candidates to minimize the maximum distance between any client and its assigned center. In this paper, we transform the p-center problem into a series of set covering subproblems, and propose a vertex weighting-based tabu search (VWTS) algorithm to solve them. The proposed VWTS algorithm integrates distinguishing features such as a vertex weighting technique and a tabu search strategy to help the search to jump out of the local optima. Computational experiments on 138 most commonly used benchmark instances show that VWTS is highly competitive comparing to the state-of-the-art methods in spite of its simplicity. As a well-known NP-hard problem which has already been studied for over half a century, it is a challenging task to break the records on these classic datasets. Yet VWTS improves the best known results for 14 out of 54 large instances, and matches the optimal results for all remaining 84 ones. In addition, the computational time taken by VWTS is much shorter than other algorithms in the literature.

1 Introduction

As a classical combinatorial optimization problem introduced by Hakimi [1964], the p-center problem consists of choosing p centers from a set of candidate centers to serve a set of clients, where each client is served by one of its closest centers. There is a serving arc if a center serves a client, and the length of the longest serving arc is the covering radius. The objective of this problem is to minimize the covering radius.

The p-center problem has a wide range of real-world applications. For instance, one can formulate several important problems in city planning as the p-center problem, such as determining the locations of emergency centers [Toregas *et al.*, 1971] and fire stations [Tansel *et al.*, 1983; Badri *et al.*, 1998]. Regarding the supply-chain management, production plant location and warehouse distribution can be modeled as the p-center problem [Amiri, 2006]. In telecommunication, the p-center problem can formulate the reliable hub network design problem [Tran *et al.*, 2016].

As a challenging NP-hard problem [Kariv and Hakimi, 1979], the p-center problem has attracted much attention from academic society during last decades. The solution methods of the p-center problem can be mainly categorized into exact algorithms and metaheuristic algorithms. Regarding the exact algorithms, Minieka [1970] converted the p-center problem into a series of set covering problems. Based on Minieka [1970], Daskin [2002] solved the p-center problem using Lagrangian relaxation. Ilhan *et al.* [2002] proposed an algorithm which finds the lower bound by optimizing the linear relaxation of the original problem, then it improves this lower bound by proving the infeasibility of a series of decision problems via integer programming. Elloumi *et al.* [2004] independently designed a similar approach as Ilhan *et al.* [2002] and obtained better results. Calik and Tansel [2013] proposed an integer programming formulation and a decomposition-based exact algorithm for tackling the p-center problem.

Apart from the exact methods, various metaheuristic algorithms have been proposed for solving the p-center problem. Mladenović *et al.* [2003] presented a tabu search and a variable neighborhood search. Caruso *et al.* [2003] proposed a hybrid algorithm which efficiently solves some small-scale p-center instances. Pullan [2008] presented a memetic algorithm which combines a local search procedure with a population-based metaheuristic algorithm. Davidović *et al.* [2011] solved the p-center problem by bee colony optimization. Ferone *et al.* [2017] proposed a GRASP-based algorithm and Yin *et al.* [2017] combined path-relinking with GRASP to tackle the p-center problem.

This paper presents a vertex weighting-based tabu search (VWTS) algorithm which incorporates a vertex weighting technique and a tabu search procedure for solving the p-center problem. Distinguished from the previous metaheuristics which directly tackle the original problem, VWTS converts the p-center problem into a series of set covering problems. Specifically, for each possible covering radius r, it checks if each client can be served by any center within the covering radius r. Our contributions can be summarized as follows:

1) We transform the p-center problem to a series of decision subproblems, each of which is further converted to a new optimization problem to reduce the complexity.

*Corresponding author

2) By combining a vertex weighting technique with a tabu search procedure, and using incremental neighborhood evaluation, VWTS reaches better balance between exploitation and exploration despite its simplicity.
3) Tested on two sets of totally 138 most commonly used instances of the p-center problem, the proposed VWTS algorithm improves the best known results on 14 instances and matches the records on the remaining ones. Moreover, our computational time is much shorter than that of the state-of-the-art algorithms in the literature.
4) We conduct extensive experiments to justify the merits of the key components of our algorithm, which are the vertex weighting technique, the tabu search strategy, and the incremental neighborhood evaluation, respectively.

2 Problem Description and Transformation

The p-center problem is defined on an undirected complete graph $G = (V, C, E)$, where $V \cup C$ is the vertex set and E is the edge set. Each vertex $v_i \in V$ corresponds to one of n clients to be served by one of the p centers chosen from the set of m candidate centers C. For each edge connecting vertices i and j, d_{ij} gives its length. The solution vector can be defined as (x, y, r), where variable x_j = 1 iff a candidate center $j \in C$ is opened, variable y_{ij} = 1 iff client $i \in V$ is served by candidate center $j \in C$, and variable $r \in R^+$ is the upper bound of the covering radius. Then, the classical mixed-integer programming (MIP) model for the p-center problem can be formulated to the following model (PC).

$$(PC) \quad \min\ r, \tag{1}$$

$$\text{s.t.} \sum_{j \in C} x_j \leq p, \tag{2}$$

$$\sum_{j \in C} y_{ij} = 1, \forall i \in V, \tag{3}$$

$$y_{ij} \leq x_j, \forall i \in V, \forall j \in C, \tag{4}$$

$$\sum_{j \in C} d_{ij} y_{ij} \leq r, \forall i \in V, \tag{5}$$

$$x_j, y_{ij} \in \{0, 1\}, r \in R^+, \forall i \in V, \forall j \in C. \tag{6}$$

In model (PC), objective (1) aims to minimize the covering radius. Constraints (2)–(5) requires that there is at most p opened centers, each client must be served by exactly one center, only the opened centers can serve the clients, and the covering radius r is not shorter than any serving arc from each client to its assigned center, respectively. It is obvious that the optimal covering radius must be the same as the length of a certain edge d_{ij}. Hence, let $\Gamma = \{r_1, r_2, ..., r_k\}$ be an ordered list of distinct edge lengths where $r_1 < r_2 < ... < r_k$, the p-center problem can be considered as seeking for the smallest rank q such that model (PC) is still feasible when $r \geq r_q$, while it becomes infeasible if constraint $r \leq r_{q-1}$ is added.

When the optimal edge length rank q of an instance is given, the p-center problem is equivalent to the set covering problem [Chvatal, 1979]. Specifically, we use a vertex set $\mathcal{V}_j^q = \{i \in V | d_{ij} \leq r_q\}$ to denote the set of clients that candidate center $j \in C$ can serve within covering radius r_q. Thus, we can obtain a set covering instance $\mathcal{V}^q = \{\mathcal{V}_1^q, \mathcal{V}_2^q, ..., \mathcal{V}_m^q\}$. If there exists p sets out of $\mathcal{V}^q$ whose union contains all vertices, we can claim that the corresponding p centers are able to serve all clients within the covering radius r_q. Therefore, the set covering problem defined by Eqs. (7)–(10) in the following model (SC_q) is equivalent to model (PC) when the optimal edge length rank q is specified, where u_i is a binary variable which is 1 if client i is not covered by any center, and x_j is the same decision variable in model (PC).

$$(SC_q) \quad \min \sum_{i \in V} u_i, \tag{7}$$

$$\text{s.t.} \sum_{j \in C, d_{ij} \leq r_q} x_j \geq 1 - u_i, \forall i \in V, \tag{8}$$

$$\sum_{j \in C} x_j = p, \tag{9}$$

$$x_j, u_i \in \{0, 1\}, \forall i \in V, \forall j \in C. \tag{10}$$

Model (SC_q) is slightly different from the standard set covering model or the MIP model presented by Elloumi *et al.* [2004]. Instead of minimizing the number of chosen sets, it minimizes the number of uncovered clients by exactly opening p centers as shown in Eq. (7). Constraints (8) ensure that for each client there is at least one center within the covering radius r. Constraint (9) limits the number of opened centers.

Unfortunately, we are usually lack of the priori knowledge about the optimal covering radius, so we need to iterate over the distinct edge length list Γ to examine each possible radius. In fact, by utilizing the bounds of the original problem, the number of subproblems can be greatly reduced.

3 Vertex Weighting-Based Tabu Search

In order to tackle the p-center problem, the proposed VWTS algorithm combines the tabu search strategy and the vertex weighting technique to solve a series of subproblems. Starting from an upper bound r_{q^0} obtained by executing solvers for model (PC) such as the PBS algorithm [Pullan, 2008] under limited time, VWTS solves models (SC_{q^0-1}), (SC_{q^0-2}), ..., (SC_1) in turn until it fails to find any feasible solution for model (SC_{q^*-1}) within the given time limit. Then, r_{q^*} is the best covering radius found. Therefore, our focus will be on the solution method for model (SC_q) with a given radius r_q.

The main framework of the proposed VWTS algorithm is presented in Algorithm 1. It generates an initial solution X by a greedy algorithm (line 1), and iteratively improves the incumbent solution by a tabu search procedure (lines 4–14). At each iteration of the VWTS algorithm, it first evaluates the neighborhood of the current solution and records the best neighborhood move while respecting their tabu states (line 5). Then, it makes the best move and replaces the current solution with the resulting neighboring solution (line 6). If the current solution X improves the best solution found so far, then X^* is updated with X (lines 7–8). Otherwise, we need to check whether the current solution falls into a local optimum, that is, the best move returned by function FindPair() cannot reduce the number of uncovered clients (line 9). If stagnation occurs, the weight of each uncovered client is adjusted (line

Algorithm 1 The main framework of the VWTS algorithm

Input: A graph G, a center number p, a covering radius r_q
Output: The best solution found so far X^*

1: A set of p centers $X \leftarrow \text{Init}(G, p, r_q)$ /* (Section 3.1) */
2: $X^* \leftarrow X$, $X' \leftarrow X$, tabu list $TL \leftarrow \varnothing$, $iter \leftarrow 1$
3: Vertex weights $w_i \leftarrow 1, \forall i \in V$ /* (Section 3.2) */
4: **while** termination condition is not met **do**
5: $(i, j) \leftarrow \text{FindPair}(X', TL, iter)$ /* (Algorithm 2) */
6: MakeMove(i, j) /* (Algorithm 4) */
7: **if** $|U(X)| < |U(X^*)|$ **then** /* $U(X)$ is the set of */
8: $X^* \leftarrow X$ /* clients uncovered by X */
9: **else if** $|U(X)| \geq |U(X')|$ **then**
10: $w_v \leftarrow w_v + 1, \forall v \in U(X)$ /* (Section 3.2) */
11: **end if** /* more uncovered clients than last solution */
12: $TL \leftarrow \{i, j\}$ /* update tabu list (Section 3.4) */
13: $X' \leftarrow X$, $iter \leftarrow iter + 1$
14: **end while**

10). Finally, when the specified termination condition is met, the algorithm terminates and returns the best solution X^*.

3.1 Initialization

The VWTS algorithm employs a constructive heuristic for generating a set of centers X as the initial solution. Let $\mathcal{V}_j$ denote the set of clients that candidate center j can serve within the current covering radius. $\mathcal{C}_i$ gives the set of candidate centers which are able to serve client i, that is, $\mathcal{C}_i = \{j \in C | i \in \mathcal{V}_j\}$. $U(X) = V \setminus \bigcup_{j \in X} \mathcal{V}_j$ represents the set of clients that are not served in solution X. The constructive heuristic opens centers one by one under a maximal coverage principle. Precisely, it iteratively selects a candidate center $j = \arg\max_{j \in C \setminus X} |\mathcal{V}_j \cap U(X)|$ which covers most uncovered clients, and inserts it into the current solution X. If there are multiple candidate centers covering the same number of uncovered clients, ties are broken randomly. The time complexity of the construction is $O(np)$.

3.2 Vertex Weighting Technique

The vertex weighting technique helps the search to escape from the local optima by altering the objective function. It can be regarded as a variant of guided local search [Voudouris and Tsang, 2003] and has been successfully applied to many problems, such as unicost set covering problem [Gao *et al.*, 2015], minimum vertex cover problem [Cai *et al.*, 2011] and satisfiability problem [Luo *et al.*, 2012]. In the p-center problem, we adapt objective (7) as Eq. (11) presents.

$$(SC_q^w) \qquad \min\ f(X) = \sum_{\forall i \in V} w_i u_i, \qquad (11)$$
$$\text{s.t. (8)–(10).}$$

Therefore, the VWTS algorithm actually works on model (SC_q^w). Note that it is a dynamic model where w_i varies as the search proceeds. If the VWTS algorithm keeps failing to cover a client, it implies that this vertex is hard to cover and we should treat it with higher priority. Specifically, when the tabu search is trapped in local optimal solution X, the VWTS algorithm increases the weight w_i of each uncovered client $i \in U(X)$ by one unit (Algorithm 1, lines 9–11). Nevertheless, the objective value of an optimal solution is always zero regardless of the configuration of the weights. This process changes the landscape of the solution space such that X is no longer a local optimum, and the search will be able to continue to explore other search areas. The more frequent a client appears in $U(X)$ when encountering stagnations, the greater its weight will be. On the one hand, the vertex weighting technique is able to prevent vertices from being repeatedly uncovered and diversify the search in an adaptive manner. On the other hand, it modifies the solution space in a smooth way, which can guide the search to promising search regions.

3.3 Neighborhood Structure and Evaluation

In order to improve the initial solution under model (SC_q^w), VWTS employs a swap-based neighborhood structure inspired by most local search-based metaheuristics for the p-center problem [Pullan, 2008], while the neighborhood evaluation is significantly different due to the reformulation presented in Section 2 and the following acceleration strategies. Denoted by Swap(i, j), a swap move produces a neighboring solution $X \oplus \text{Swap}(i, j) = X \cup \{i\} \setminus \{j\}$, by opening a candidate $i \in C \setminus X$, and closing an opened center $j \in X$.

The neighborhood evaluation is the most time consuming procedure in a trajectory-based metaheuristic algorithm. For a typical tabu search algorithm which adopts the best improvement policy, it evaluates all feasible moves at each iteration, and performs one of the best neighborhood moves which improves the objective value as much as possible. Since there are $O(p(n-p))$ swap moves, the size of the neighborhood could be huge on some large instances. Therefore, we use a neighborhood sampling strategy and an incremental evaluation technique to accelerate the evaluation.

On the one hand, the objective value can only be improved by covering some uncovered clients, so the VWTS algorithm will only evaluate a swap move Swap(i, j) if i covers some uncovered vertices in $U(X)$. Since each client must be eventually covered, the VWTS algorithm randomly picks a vertex $k \in U(X)$, and only evaluates neighborhood moves Swap(i, j) where $i \in \mathcal{C}_k$ and $j \in X$. Apart from the reduced time consumption for the neighborhood evaluation, the diversification of the search is improved as a side effect, reaching a better balance between intensification and diversification.

On the other hand, we try to accelerate the neighborhood evaluation by reusing some intermediate results for calculating the objective values. Instead of naively summing the weights of the covered clients according to objective (11), VWTS incrementally evaluates all neighborhood moves by storing and maintaining δ_j, the objective value increase (decrease) by closing (opening) center j. For each center $j \in X$, $\delta_j = \sum_{i \in (\mathcal{V}_j \cap U(X \setminus \{j\}))} w_i$ is the sum of the weights of the vertices which can only be served by center j. For each candidate center $j \notin X$, $\delta_j = \sum_{i \in (\mathcal{V}_j \cap U(X))} w_i$ is the sum of the weights of all uncovered clients in $\mathcal{V}_j$. Then, the evaluation for each neighborhood move can be implemented in $O(1)$ time complexity, at the cost of updating the affected δ values every time after opening or closing a center. Specifically, it calculates the objective value for opening center i by $f(X \cup \{i\}) = f(X) - \delta_i$. Then, we need to update the

Algorithm 2 Find the best swap pair

```
1: function FindPair(X, TL, iter)
2:     The set of best swap moves M ← ∅
3:     The best objective value obj ← +∞
4:     v ← a randomly picked uncovered vertex in U(X)
5:     δ'_j ← δ_j, ∀j ∈ C        /* backup before trial moves */
6:     for all i ∈ C_v do        /* C_v: candidates covering v */
7:         TryToOpenCenter(i)            /* (Algorithm 3) */
8:         for all j ∈ X do    /* evaluate closing center j */
9:             if {i, j} ∩ TL = ∅ then  /* not tabu move */
10:                if f(X ⊕ Swap(i, j)) < obj then
11:                    obj ← f(X ⊕ Swap(i, j))
12:                    M ← {Swap(i, j)}
13:                else if f(X ⊕ Swap(i, j)) = obj then
14:                    M ← M ∪ {Swap(i, j)}
15:                end if
16:            end if
17:        end for
18:        δ_j ← δ'_j, ∀j ∈ C     /* restore after trial moves */
19:    end for                  /* v ∈ U(X) ⇔ C_v ∩ X = ∅ */
20:    return a randomly picked move in M
21: end function
```

Algorithm 3 Open a center virtually

```
1: function TryToOpenCenter(i)
2:     for all v ∈ V_i do   /* |X ∩ C_v|: number of centers */
3:         if |X ∩ C_v| = 1 then          /* covering v in X */
4:             /* cancel penalty for making v uncovered */
5:             δ_l ← δ_l − w_v, for l ∈ X ∩ C_v        /* O(1) */
6:         end if   /* l was the only center covering v but */
7:     end for  /* it will not be the only one if i opens so */
8: end function  /* closing l does not make v uncovered */
```

δ values affected by opening center i which eliminates the penalties for closing some centers due to the overlapped coverage. After that, we can evaluate the neighboring solutions by $f(X \oplus \text{Swap}(i,j)) = f(X \cup \{i\}) + \delta_j$. As tabu search procedure evaluates a large number of moves but only makes a single move at each iteration, it is worthy of maintaining and querying the cache rather than computing the objective function from scratch for every swap move.

Algorithm 2 describes the neighborhood evaluation procedure. It randomly selects a client $k \in U(X)$ (line 4) and tries to open each candidate center which covers vertex k (lines 6–7). The sub-routine TryToOpenCenter(i) keeps each δ_j up-to-date to accelerate the calculation of the objective function $f(X \oplus \text{Swap}(i,j))$. For each non-tabu neighboring solution, if its objective value is smaller than obj, the corresponding neighborhood move is saved as the best move (lines 9–16). When all trial moves regarding opening candidate center i are evaluated, we need to restore the δ values (line 18).

Apparently, covering a client v which has already been covered ($|X \cap \mathcal{C}_v| \geq 1$) will not improve the objective value, and closing the center which serves a client v will not worsen the objective value if there are multiple centers which can cover vertex v ($|X \cap \mathcal{C}_v| \geq 2$). Therefore, as presented in Algorithm 3, if there is exactly one center l which covers client v ($|X \cap \mathcal{C}_v| = 1$) before opening center i, then the δ value for closing center l which used to make client v uncovered will decrease by w_v (lines 4–7), because closing center l will not make client v uncovered once center i is opened. We are only interested in $\delta_j (\forall j \in X)$ for closing a center, since lines 8–17 in Algorithm 2 only concern closing another center. As a result, the worst-case time complexity of Algorithm 2 and Algorithm 3 are $O(n^2)$ and $O(n)$, respectively.

Algorithm 4 Make a swap move

```
1: function MakeMove(i, j)
2:     for all v ∈ V_i do      /* consequences of opening i */
3:         if |X ∩ C_v| = 1 then          /* (Algorithm 3) */
4:             δ_l ← δ_l − w_v, for l ∈ X ∩ C_v
5:         else if |X ∩ C_v| = 0 then
6:             δ_l ← δ_l − w_v, ∀l ∈ C_v \ {i}
7:         end if          /* cancel reward for covering v */
8:     end for
9:     X ← X ∪ {i} \ {j}
10:    for all v ∈ V_j do      /* consequences of closing j */
11:        if |X ∩ C_v| = 0 then          /* add reward for */
12:            δ_l ← δ_l + w_v, ∀l ∈ C_v \ {j}  /* covering v */
13:        else if |X ∩ C_v| = 1 then
14:            δ_l ← δ_l + w_v, for l ∈ X ∩ C_v
15:        end if          /* add penalty for uncovering v */
16:    end for
17: end function
```

When the best swap move is performed (Algorithm 1, line 6), we need to update the affected data structures by calling Algorithm 4. In addition to updating the δ values as Algorithm 3 (lines 3–4), if there is no center covering client v ($|X \cap \mathcal{C}_v| = 0$) before opening center i, then the δ value for opening each candidate center l to cover client v is decreased by w_v (lines 5–7), because opening center l to cover the already covered client v in the future will no longer improve the objective value by w_v. Then, it updates the set of opened centers (line 9). The influences on δ values for closing center j are taken into account in a similar manner (lines 10–16).

3.4 Tabu Search

Tabu search usually incorporates a recency-based tabu list to prohibit revisiting recently visited solutions. The tabu strategy prevents closing newly opened centers or reopening newly closed ones immediately. We fix the parameter of the tabu tenure in the tabu strategy to one iteration, so the proposed algorithm keeps simple and parameter-free. Specifically, if we open (close) a center at the current iteration $iter$, it is forbidden to close (open) it again at the the next iteration. Thus, Swap(i, j) at iteration $iter$ introduces two vertices $\{i, j\}$ in tabu list TL (Algorithm 1, line 12) and neither i nor j can be involved at iteration $iter + 1$ (Algorithm 2, line 9).

4 Experimental Results and Comparisons

In order to evaluate the effectiveness of the proposed VWTS algorithm, we conduct extensive experiments on the well-known datasets, and compare VWTS with the state-of-the-

art algorithms including two exact algorithms (ELP [Elloumi *et al.*, 2004] and DBR2 [Calik and Tansel, 2013]) and three metaheuristic algorithms (PBS [Pullan, 2008], GRASP+PS [Ferone *et al.*, 2017], and GRASP/PR [Yin *et al.*, 2017]).

4.1 Experimental Protocol

Our proposed VWTS algorithm is programmed in C++ and compiled with Visual Studio 2017. All experiments are carried out on Windows Server 2012 x64 with Intel Xeon E5-2609v2 2.50 GHz CPU. We run a reimplemented PBS algorithm [Pullan, 2008] under a 5-millisecond time limit to obtain a good initial upper bound of the covering radius r_{q^0} for each instance. We performed 20 independent runs on each instance, and the cutoff time for each subproblem induced by each covering radius is 6 minutes. Since the programs of the reference algorithms are not available, we report their computational time in the corresponding papers and normalize the run time according to CPU frequency for a fair comparison. The run time for ELP is divided by 6.25 for it runs on a 400 MHz Pentium II CPU. The remaining algorithms are tested on faster (PBS, GRASP+PS, GRASP/PR) or unknown (DBR2) CPUs, so we report their computational time as is.

There are mainly two sets of benchmark instances for the p-center problem. The first set consists of 40 small instances (pmed) from OR-Library [Beasley, 1990]. The second set contains 98 instances from TSPLIB which are based on planar graphs and derived from real-world applications [Reinelt, 1991]. The TSPLIB dataset can be further divided into 44 small instances (sTSP) with less than 1000 vertices, and 54 large ones (u1060, rl1323, u1817, pcb3038). A client is also a candidate center ($V = C$) in these instances. We compute the all-pair shortest paths by Floyd algorithm [Floyd, 1962] for the pmed instances, and round the Euclidean distances to the nearest hundredths for the TSPLIB instances, as Pullan [2008] did. Note that algorithm ELP and DBR2 adopt integer distances which are too rough to figure out the optimal q^*.

4.2 Computational Results

In this section, we provide comprehensive investigations on the performance of our VWTS algorithm and report the detailed results in Tables 1 and 2. They follow the same convention described below. Columns Dataset and Instance give the names of datasets and instances, respectively. Columns n and p indicate the numbers of vertices and centers, respectively. Column Count presents the number of instances in the dataset. Column CPU gives the normalized average CPU time in seconds. For VWTS, it includes the total time for computing r_{q^0} and sequentially solving all subproblems from (SC_{q^0}) to (SC_{q^*}). Column CPU-q^* reports the run time in seconds for VWTS to solve the subproblem induced by the best known covering radius r_{q^*}, which shows the limit of VWTS when parallel computing is available. Column f_{best} presents the best objective values obtained by the corresponding algorithms. Column #best shows the number of instances where the corresponding algorithms match the best known results. The #better, #equal, and #worse are the number of instances where VWTS obtains better, equal, and worse results comparing to the corresponding algorithms, respectively.

Table 1 compares the overall results on each dataset obtained by ELP, DBR2, PBS, GRASP/PR, and our VWTS. We can observe that VWTS obtains the best known results for all the instances, while no reference algorithm can obtain this overall outcome. Moreover, the average computational time of VWTS on each dataset is over 4 times shorter than the best algorithms in the literature. If parallel computing is available, the performance gain can be over 30 times. The time difference is so big that the comparison is not biased by the normalization. In addition, VWTS is very stable since it always obtains the reported results in all 20 independent runs.

Table 2 compares the experimental results produced by ELP, DBR2, PBS, GRASP+PS, GRASP/PR, and our VWTS on 30 largest and most challenging instances. Note that the real upper bounds for dataset pcb3038 have not been updated for decades (although DBR2 improved some integer bounds). From Table 2 we can observe that, the proposed VWTS algorithm improves the previous best known results obtained by GRASP/PR and PBS on rl1323 with $p = 100$, u1817 with $p = 80, 130, 150$, and pcb3038 with $p \geq 50$. Apart from the improved solution quality, VWTS outperforms all reference algorithms on u1817 and pcb3038 datasets in terms of the computational time. Regarding the run time for the subproblem corresponding to the optimal or best known covering radius, VWTS is able to obtain the new best known results on 23 out of 30 instances within 3 seconds. Furthermore, if VWTS stops at the previous best covering radius, it only takes 3 seconds on 28 instances. In sum, these statistics show that the our VWTS is highly effective, efficient, robust, and concurrency-friendly for solving the p-center problem.

4.3 Importance of VWTS Ingredients

In order to evaluate the merits of the vertex weighting technique, the tabu search strategy, and the incremental neighborhood evaluation, we conducted experiments on four largest instances (pcb3038, $p = 350, 400, 450, 500$) to compare VWTS with its simplified versions obtained by disabling the tabu strategy (VW), deactivating the vertex weighting technique (TS), and using the naive neighborhood evaluation instead of the incremental one (VWTS′), respectively.

Figure 1 depicts the evolution of the uncovered vertex numbers as the search proceeds by VWTS, VW, TS, and VWTS′. Each point (x, y) means that there are $10^y - 1$ clients which are not served by any center at 10^x microsecond. We can observe that, although TS can obtain better solutions than VWTS at first milliseconds, VWTS and VW overtake after 0.1 seconds. The reason for this phenomena might be that, the tabu search pays too much attention on prohibiting a candidate center from being centers, instead of directly preventing a client from being uncovered. In extreme conditions, some hard-to-serve vertices will never get a chance to be served at all. However, the tabu strategy accelerates the convergence of VWTS. According to Figure 1, VWTS is over $10^{0.2} = 1.5$ times faster than VW for finding a feasible covering. Moreover, when the incremental evaluation is disabled, VWTS′ is 100 times slower than the complete algorithm. These observations confirm that the vertex weighting technique, the tabu strategy, and the incremental evaluation are all essential for VWTS in terms of effectiveness and efficiency.

Dataset	Count	ELP			DBR2			PBS			GRASP/PR			VWTS		
		#best	#better	CPU	#best	#better	CPU	#best	#better	CPU	#best	#better	CPU	#best	CPU-q^*	CPU
pmed	40	*40*	0	0.75	*40*	0	0.65	*40*	0	2.91	*40*	0	0.12	*40*	<0.01	<0.01
sTSP	44	-	-	-	-	-	-	42	2	12.38	*44*	0	1.02	*44*	<0.01	<0.01
u1060	15	15	0	4.54	-	-	-	12	3	151.13	*15*	0	90.87	*15*	0.03	0.47
rl1323	10	10	0	10.35	-	-	-	9	1	3464.29	9	1	1898.46	**10**	0.19	1.80
u1817	15	8	7	38.02	15	0	675.69	7	8	2792.08	12	3	1286.03	**15**	0.86	2.35
pcb3038	14	-	-	-	5	5	33066.00	4	10	1000.18	4	10	3564.54	**14**	33.36	212.91

Table 1: Average computational results on all six datasets. Numbers in bold (italic) indicate the improved (matched) best known results.

Instance	n	p	ELP		DBR2		GRASP+PS	PBS		GRASP/PR		VWTS		
			f_{best}	CPU	f_{best}	CPU	f_{best}	f_{best}	CPU	f_{best}	CPU	f_{best}	CPU-q^*	CPU
rl1323	1323	100	*787*	4.16	-	-	886.85	789.70	1,840.50	789.70	2010.67	**787.10***	0.31	1.26
u1817	1817	10	*458*	97.76	*458*	22.11	466.96	*457.91*	5316.60	*457.91*	604.53	*457.91**	1.85	1.78
u1817	1817	20	310	105.60	*309*	278.49	330.20	*309.01*	10243.00	*309.01*	4068.06	*309.01**	2.92	8.56
u1817	1817	30	250	56.80	*241*	344.59	265.19	*240.99*	1605.50	*240.99*	1239.97	*240.99**	1.52	5.23
u1817	1817	40	210	39.52	*209*	1221.86	232.25	209.46	193.70	*209.45*	308.29	*209.45**	0.91	1.41
u1817	1817	50	187	38.72	*185*	330.09	204.79	*184.91*	1128.90	*184.91*	471.94	*184.91**	0.08	1.20
u1817	1817	60	*163*	28.32	*163*	17.52	184.91	162.65	837.30	*162.64*	469.43	*162.64**	0.07	0.93
u1817	1817	70	*148*	26.56	*148*	6.04	170.39	*148.11*	191.80	*148.11*	19.66	*148.11**	0.05	0.45
u1817	1817	80	*137*	24.00	*137*	35.12	154.50	136.80	127.50	136.80	12.42	**136.77***	3.00	1.66
u1817	1817	90	*130*	25.76	129(?)	7519.04	148.11	129.54	2963.50	*129.51*	3859.05	*129.51**	0.55	0.59
u1817	1817	100	*127*	25.44	*127*	10.22	136.79	127.01	146.40	*126.99*	2.35	*126.99**	0.02	0.27
u1817	1817	110	*109*	19.04	110	5.32	-	*109.25*	13772.40	*109.25*	6954.89	*109.25**	0.27	1.70
u1817	1817	120	*108*	20.96	107(?)	3.99	-	107.78	80.10	*107.76*	5.25	*107.76**	0.02	0.25
u1817	1817	130	108	19.36	105	335.03	-	107.75	11.20	107.75	7.04	**104.73***	1.20	9.17
u1817	1817	140	105	19.36	*102*	4.62	-	101.61	4949.30	*101.60*	30.95	*101.60**	0.39	0.60
u1817	1817	150	94	23.04	92	6.61	-	101.60	314.00	92.44	1236.55	**91.60***	0.07	1.41
pcb3038	3038	10	-	-	*729*	176.23	-	*728.54*	5415.40	*728.54*	240.56	*728.54*	0.53	23.37
pcb3038	3038	20	-	-	*493*	22740.76	-	*493.04*	972.10	*493.04*	1051.99	*493.04*	1.74	38.28
pcb3038	3038	30	-	-	397	76923.67	-	*393.50*	438.60	*393.50*	644.34	*393.50*	1.23	69.22
pcb3038	3038	40	-	-	337	72364.56	-	*336.42*	763.00	*336.42*	420.28	*336.42*	79.81	264.98
pcb3038	3038	50	-	-	300	91029.77	534.48	298.20	731.60	298.10	4686.77	**298.04**	78.34	307.51
pcb3038	3038	100	-	-	209	27292.39	399.49	206.63	565.80	207.06	6678.44	**206.60**	97.85	683.89
pcb3038	3038	150	-	-	-	-	331.62	164.77	427.00	165.00	5653.32	**164.55**	88.55	705.74
pcb3038	3038	200	-	-	141	33929.91	301.01	140.90	899.40	140.62	5021.13	**140.09**	59.09	360.98
pcb3038	3038	250	-	-	-	-	292.48	122.78	723.70	122.78	1985.10	**122.25**	24.37	205.55
pcb3038	3038	300	-	-	115	6185.96	261.28	115.25	537.40	115.73	3671.32	**115.00**	29.73	133.26
pcb3038	3038	350	-	-	-	-	258.82	104.81	805.90	104.81	3926.83	**104.68**	1.64	49.33
pcb3038	3038	400	-	-	97	11.48	249.78	97.51	620.60	97.80	6956.46	**96.88**	2.89	55.58
pcb3038	3038	450	-	-	-	-	214.97	88.96	666.20	89.56	6161.75	**88.55**	0.51	49.46
pcb3038	3038	500	-	-	85	5.23	209.35	85.00	435.80	85.09	3015.45	**84.58**	0.76	33.60
#better/#equal/#worse			7/9/0		6/17/0		21/0/0	20/10/0		14/16/0				

Table 2: Computational results on the most challenging instances. Smaller f_{best} is better. "*" means the optimal objective value proven by model (SC_q). The results marked with "(?)" are incorrect according to the optimality proof. Other notations are the same as Table 1.

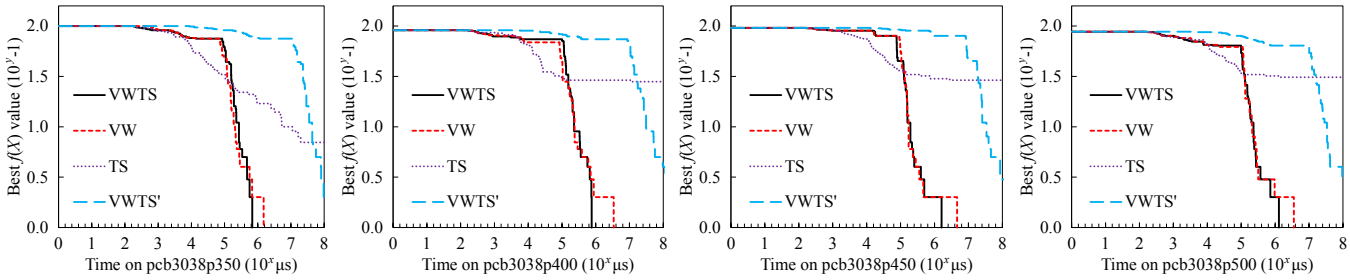

Figure 1: Evolution of the uncovered vertex number by VWTS, VW, TS, and VWTS′ on four largest instances.

5 Conclusion

This paper presented a vertex weighting-based tabu search algorithm to solve the p-center problem. We decompose the p-center problem into a series of set covering subproblems, and solve them by combining the vertex weighting technique and the tabu search strategy. Tested on 138 widely used datasets, VWTS improves the best known results on 14 instances and matches the records in the literature for all remaining ones within less run time. Thus, it would be appealing to investigate the combination of the proposed strategies for solving other optimization problems in the future.

Acknowledgments

Research partially supported by Alkaid lab of Huawei Cloud.

References

[Amiri, 2006] Ali Amiri. Designing a distribution network in a supply chain system: Formulation and efficient solution procedure. *European Journal of Operational Research*, 171(2):567–576, 2006.

[Badri *et al.*, 1998] Masood A Badri, Amr K Mortagy, and Colonel Ali Alsayed. A multi-objective model for locating fire stations. *European Journal of Operational Research*, 110(2):243–260, 1998.

[Beasley, 1990] John E Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10):1672–1696, 2011.

[Calik and Tansel, 2013] Hatice Calik and Barbaros C Tansel. Double bound method for solving the p-center location problem. *Computers & Operations Research*, 40(12):2991–2999, 2013.

[Caruso *et al.*, 2003] C Caruso, A Colorni, and L Aloi. Dominant, an algorithm for the p-center problem. *European Journal of Operational Research*, 149(1):53–64, 2003.

[Chvatal, 1979] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[Daskin, 2002] Mark S Daskin. A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. *Communications of the Operations Research Society of Japan*, 45:428–436, 2002.

[Davidović *et al.*, 2011] Tatjana Davidović, Dušan Ramljak, Milica Šelmić, and Dušan Teodorović. Bee colony optimization for the p-center problem. *Computers & Operations Research*, 38(10):1367–1376, 2011.

[Elloumi *et al.*, 2004] Sourour Elloumi, Martine Labbé, and Yves Pochet. A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing*, 16(1):84–94, 2004.

[Ferone *et al.*, 2017] Daniele Ferone, Paola Festa, Antonio Napoletano, and Mauricio GC Resende. A new local search for the p-center problem based on the critical vertex concept. In *International Conference on Learning and Intelligent Optimization*, pages 79–92. Springer, 2017.

[Floyd, 1962] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[Gao *et al.*, 2015] Chao Gao, Xin Yao, Thomas Weise, and Jinlong Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.

[Hakimi, 1964] S Louis Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.

[Ilhan *et al.*, 2002] T Ilhan, F Ozsoy, and M Pinar. An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems. Technical report, Bilkent University, Department of Industrial Engineering, 2002.

[Kariv and Hakimi, 1979] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[Luo *et al.*, 2012] Chuan Luo, Kaile Su, and Shaowei Cai. Improving local search for random 3-SAT using quantitative configuration checking. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 570–575. IOS Press, 2012.

[Minieka, 1970] Edward Minieka. The m-center problem. *SIAM Review*, 12(1):138–139, 1970.

[Mladenović *et al.*, 2003] Nenad Mladenović, Martine Labbé, and Pierre Hansen. Solving the p-center problem with tabu search and variable neighborhood search. *Networks: An International Journal*, 42(1):48–64, 2003.

[Pullan, 2008] Wayne Pullan. A memetic genetic algorithm for the vertex p-center problem. *Evolutionary Computation*, 16(3):417–436, 2008.

[Reinelt, 1991] Gerhard Reinelt. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[Tansel *et al.*, 1983] B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: A survey. *Management Science*, 29:482–511, 1983.

[Toregas *et al.*, 1971] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(6):1363–1373, 1971.

[Tran *et al.*, 2016] Trung Hieu Tran, Jesse R O' Hanley, and M Paola Scaparra. Reliable hub network design: Formulation and solution techniques. *Transportation Science*, 51(1):358–375, 2016.

[Voudouris and Tsang, 2003] Christos Voudouris and Edward P. K." Tsang. *Guided Local Search*, pages 185–218. Springer US, Boston, MA, 2003.

[Yin *et al.*, 2017] Ai-Hua Yin, Tao-Qing Zhou, Jun-Wen Ding, Qing-Jie Zhao, and Zhi-Peng Lv. Greedy randomized adaptive search procedure with path-relinking for the vertex p-center problem. *Journal of Computer Science and Technology*, 32(6):1319–1334, 2017.