Parallel iterative solution-based tabu search for the obnoxious p-median problem

Jian Chang[a], Lifang Wang[a], Jin-Kao Hao[b,c], Yang Wang[a,*]

[a] *School of Management, Northwestern Polytechnical University, 127 Youyi West Road, 710072 Xi'an, China*
[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[c] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

Accepted to Computers & Operations Research, November 2020

Abstract

The obnoxious p-median problem (OpM) is to determine a set of opened facilities such that the sum of distances between each client and the opened facilities is maximized. OpM is a general model that has a wide range of practical applications. However the problem is computationally challenging because it is known to be NP-hard. In this work, we propose an effective parallel iterative solution-based tabu search algorithm to solve OpM. The proposed algorithm combines a delete-add compound move instead of a typical time-consuming swap move to improve neighborhood exploration, a solution-based tabu search procedure to strictly prevent visited solutions from being revisited, a perturbation scheme similar to the shaking phase of variable neighborhood search for search diversification, and a parallel strategy of leveraging multiple processors of a computer. Experimental results on 144 benchmark instances demonstrate that the proposed algorithm is able to find new lower bounds for 7 instances and match the best known results for the other instances. Further experimental analysis sheds light on the key ingredients to the performance of the proposed algorithm. The code of our PISTS algorithm can be accessed on GitHub at https://github.com/changjian-github/PISTS-for-OpM, which will facilitate future comparative studies.

Keywords: Combinatorial optimization; Obnoxious p-median problem; Tabu search; Heuristics.

*Corresponding author.
Email addresses: jianchang_research@outlook.com (Jian Chang), lifang@nwpu.edu.cn (Lifang Wang), jin-Kao.hao@univ-angers.fr (Jin-Kao Hao), sparkle.wy@gmail.com (Yang Wang)

1. Introduction

Obnoxious facility location problems belong to a set of location models in which clients try to avoid facilities and keep away from them [1, 2, 3, 4, 5]. Typical applications include optimal locations of nuclear reactors, garbage dumps, electric power supplier networks, water purification plants, etc [6, 7, 8, 9]. Take the locations of quarantine sites as an example. In the epidemic situation, a large number of quarantine sites need to be established for patient admission. Locations of such public facilities are required to be far away from the residential communities for the reason of potential infection. Due to its wide applications, various models and solution approaches have been proposed in the literature [10, 11, 12, 13, 14].

In this paper, we focus on the well-known obnoxious p-median problem (OpM), the objective of which is to determine a set of opened facilities such that the sum of distances between each client and the opened facilities is maximized. The distance between a client and the opened facilities is defined as the minimum distance between this client and each opened facility. For solving OpM, a branch-and-cut (B&C) algorithm and several heuristic algorithms have been proposed in the literature [4, 15, 16, 17, 18]. The B&C algorithm is experimentally demonstrated to work well for solving medium instances. Due to the NP-hard nature of the problem [19], heuristic algorithms are more widely studied to solve large and challenging OpM instances.

Colmenar et al. [4] developed a greedy randomized adaptive search procedure (GRASP), by investigating different greedy constructive and local search procedures. The constructive procedure either randomly chooses a solution from a candidate list constructed by a greedy strategy or chooses a solution in a greedy way from a set of randomly constructed solutions. Based on the first improvement strategy, the local search procedures swap elements either randomly or based on the objective contribution. Extensive experiments show that the proposed GRASP algorithm outperforms the B&C algorithm of [15] by reducing the computational time geometrically.

Lin et al. [16] proposed a hybrid binary particle swarm optimization algorithm (HBPSO) that combines typical particle swarm optimization (PSO) search with greedy iterative local search. An innovative add-drop operation is incorporated that greedily adds an element into a solution and then greedily drops an element out of the resulting partial solution. Experimental results disclose that new better solutions can be found within shorter computational time compared to GRASP.

Herrán et al. [17] designed a parallel algorithm based on variable neighborhood search (PVNS) for solving OpM. The local search procedure sequentially adds and drops an element in a greedy way, in order to quickly obtain an approximation of the best solution in the typical swap neighborhood. Moreover, multiple sequential VNS algorithms are simultaneously run to make full use of the processors in the computer. Extensive experiments reveal that this PVNS approach performs well in terms of solution quality and computational efficacy.

Mladenović et al. [18] developed a basic variable neighborhood search (VNS)

algorithm that consists of a random initial solution constructive procedure, a local search procedure and a random interchange move based shaking procedure. The local search procedure identifies the best facility to replace a given facility rather than the best interchange among all the feasible pairs of facilities as in the typical best improvement search strategy. Moreover, a heap data structure is incorporated to efficiently evaluate the neighbor solutions. Although the proposed algorithm is simple in nature, it is experimentally demonstrated to be competitive with other existing OpM algorithms.

We observe that most heuristic algorithms proposed for solving OpM employ a basic local search procedure based on the typical swap neighborhood, which is time-consuming to identify the best solution at each local search iteration. Meanwhile, as a popular and powerful method, tabu search enhances the performance of local search techniques by prohibiting previously visited solutions from being revisited [20]. In particular, the solution-based tabu list is able to eliminate the need of tuning the tabu tenure that typically impacts the search performance [21]. Hence, this invites us to design the solution-based tabu search for performing neighborhood exploration. Furthermore, an extension of the standard tabu search strategy, known as iterated tabu search, incorporates a diversification mechanism to overcome the difficulty of the search trajectory being confined in a "narrow region" of the solution space. In addition, multi-core processors are now available in almost all personal computers and algorithms that exploit parallel techniques on such a computer can reduce their computation time. For instance, the parallel VNS algorithm proposed in [17] is among the best performing OpM methods. Yet, neither iterated tabu search nor its parallel implementation was investigated for solving OpM in the literature.

In this work, we introduce the first parallel iterative solution-based tabu search (PISTS) algorithm that alternates between a solution-based tabu search procedure for search intensification and an adaptive perturbation procedure for search diversification. The original features of PISTS and the contributions of this work are summarized as follows.

First, we analyze different compound moves to improve the neighborhood exploration and experimentally determine that the 'delete-add' compound move is a good alternative to the typical time-consuming 'swap' move. To further improve tabu search, we use two separate tabu lists to forbid incomplete solutions obtained by performing a 'delete' move and the feasible solutions obtained by performing an 'add' move. This has the merit of exploring a wider area within the given time limit by strictly preventing solutions from being revisited.

Second, the adaptive perturbation procedure is inspired by the shaking phase in the VNS algorithm to dynamically adjust the perturbation strength k, which systematically increases the distance between the new starting solutions from the best found solution when the search stagnates and decreases to the minimum distance when the search is improving. The perturbation procedure is randomized in nature by performing k random 'delete-add' moves to achieve search diversification.

Third, experimental testings on 144 benchmark instances in the literature indicate that the PISTS algorithm competes favorably with the state-of-the-art

algorithms by establishing new lower bounds for 7 instances and matching the best known results for all the other instances.

Finally, the proposed compound move combined with the use of two different tabu lists is a general intensification strategy and can advantageously be applied to other cardinality constrained binary optimization problems.

The rest of this paper is organized as follows. Section 2 introduces the problem formulation and neighborhood definition. Section 3 presents the main scheme and important components of the proposed algorithm. Section 4 exposes preliminary experiments to analyze the important search ingredients. Section 5 reports computational results and comparisons with state-of-the-art algorithms in the literature. Section 6 draws conclusions and suggests future research directions.

2. Preliminaries

2.1. Problem formulation

Let I denote a set of clients, J denote a set of facilities, d_{ij} be the distance between client i and facility j. The objective of OpM is to find p opened facilities (p is given), i.e. a subset $S_p = \{j_1, j_2, \ldots, j_p\} \subset J$ satisfying

$$\max_{S_p} f = \sum_{i \in I} \min \{d_{ij} : j \in S_p\} \tag{1}$$

In Equation (1), $\min\{d_{ij} : j \in S_p\}$ denotes the distance between a given client i and all the opened facilities, defined to be the minimum value among all the distances between i and each open facility j. The objective function f is to maximize the sum of the distances between all clients and the open facilities.

2.2. Neighborhood definition

Distance between solutions: For a given solution S_p, its nearest neighbor S'_p differs only in one element from S_p, obtained by swapping an element in S_p and an element in $J \backslash S_p$. Alternatively, we define the distance between S_p and S'_p as

$$dist(S_p, S'_p) = |S_p \backslash S'_p| \tag{2}$$

where $|\cdot|$ denotes the cardinality of the set. Thus, the distance between a solution and its nearest neighbor is 1.

Neighborhood N_k: The neighborhood N_k of a solution S_p consists of all the solutions having a distance of k from the solution S_p, which is formulated as

$$N_k = \{S'_p : dist(S_p, S'_p) = k, S'_p \in \Omega_p\} \tag{3}$$

where Ω_p denotes the solution space. The nearest neighborhood N_1 results when $k = 1$.

3. Parallel iterative solution-based tabu search

3.1. General scheme

The general scheme of the proposed parallel iterative solution-based tabu search (PISTS) is shown in Algorithm 1. The parallel implementation employs the multiprocessing module of Python to leverage multiple processors on a given machine[1]. The PISTS algorithm first creates an object $ProcPool$ of the class $Pool$ to store n_{proc} processes. A shared memory mechanism is introduced so that all processes can get access to the tabu lists. In this way, the solutions previously visited by a process will not be revisited by the other processes. Preliminary experiments indicate that such a parallel implementation requires much less computational time to reach the same solutions compared to the typical parallel implementation without information interaction among processes, especially for solving instances of large size. Then it repeats creating each process by including a random initializer used for generating a randomized initial solution and an iterative solution-based tabu search procedure $ISTS()$ used for improving the initial solution. After all the specified n_{proc} processes are completed, $ProcPool$ is run to execute each process on a processor and the best solution S_{best} aggregated from all the processes is returned.

Algorithm 1 General scheme of PISTS

Input: number of processes n_{proc}, time limit T, tabu search depth d, iteration cutoff h of using a specific neighborhood N_k, tabu lists $TabuListADD$ and $TabuListDLT$
Output: best solution S_{best}
Create an object $ProcPool = Pool()$ to store n_{proc} processes
Initialize tabu lists $TabuListDTL$ and $TabuListADD$ in shared memory
for $i = 1$ to n_{proc} **do**
    $S_i = random\_initializer()$
    Add $ISTS(S_i, d, h, T_g, TabuListADD, TabuListDTL)$ to $ProcPool$
end for
Run $ProcPool$ to return the best solution S_{best}

3.2. Iterative solution-based tabu search

Our proposed iterative solution-based tabu search (ISTS) shown in Algorithm 2 alternates between a solution-based tabu search procedure and a perturbation procedure similar to the shaking phase in the VNS algorithm [17, 22]. For a given initial solution, the solution-based tabu search procedure is first executed to reach a local optimum. Unlike the common tabu strategy which typically forbids previously performed *moves* for a period of time, our solution-based tabu strategy prevents the previously visited solutions from being revisited. The perturbation procedure dynamically adjusts the perturbation strength

[1] https://docs.python.org/3/library/multiprocessing.html

k, which systematically increases the distance between the new starting solutions from the best found solution when the search stagnates and decreases to the minimum distance when the search is improving.

The outer 'while' loop of Algorithm 2 keeps exploring a sequence of N_k neighborhoods until a given time limit T is met. For each specific neighborhood N_k, the inner 'while' loop iteratively executes the perturbation procedure to generate a new solution S_p and the solution-based tabu search procedure to reach a new local optimum S_l by refining the quality of S_p. Given that the perturbation procedure employs a random strategy that performs a number of k random delete-add moves, each N_k neighborhood is iteratively explored until the best solution is not improved for consequtive h number of iterations (h is called N_k iteration cutoff). Moreover, this limited number of iterations used for exploring a specific neighborhood provides chances for exploring other regions to avoid getting trapped in local optimum.

Algorithm 2 General scheme of ISTS

```
Input: initial solution S_i, tabu search depth d, N_k iteration cutoff h, time limit T, tabu lists TabuListADD and TabuListDLT
Output: best solution found during the search S_g
Initialize tabu lists TabuListADD = ∅ for solutions obtained by performing add moves and TabuListDLT = ∅ for solutions obtained by performing delete moves, k = 2, t_0 = time()
S_g = STS(S_i, d, TabuListADD, TabuListDLT)  ▷ STS() is defined in Algorithm 3
while time() − t_0 < T do                    ▷ Iterative search in varying N_k
    nonimp = 0                               ▷ Iterative search in a specific N_k
    while TRUE do
        S_p = perturb(S_g, k)
        S_l = STS(S_p, d, TabuListADD, TabuListDLT)
        if f(S_l) > f(S_g) then
            S_g = S_l, k = 2, nonimp = 0
            break
        else
            nonimp = nonimp + 1
        end if
        if nonimp > h then
            k = k + 1
            break
        end if
    end while
    if k > p or k > n − p then
        k = 2
    end if
end while
```

ISTS starts with $k = 2$. If the best solution S_g is not improved for a consequtive h number of iterations when exploring a specific neighborhood N_k,

k is increased by 1 to switch the search to neighborhood N_{k+1}. As long as a new best solution S_g is found, the search returns to neighborhood N_2. Since for any solution, at most $min\{p, |J| - p\}$ elements can be deleted and added, i.e., the largest jump distance is $min\{p, |J| - p\}$, k is also reset to 2 once $k > min\{p, |J| - p\}$ to guarantee the feasibility of the perturbed solution.

3.3. Solution-based tabu search

The solution-based tabu search (STS) procedure improves a starting solution to reach a new local optimum. In order to design an effective tabu search, two important issues should be considered.

The first one is to design a method that is able to quickly identify the best non-tabu move for each iteration. Swap neighborhood is generally adopted for solving problems with the similar solution structure [23]. For solving OpM, a traditional swap neighborhood leads to $p(|J| - p)$ neighbor moves. Hence, it is time-consuming to identify the best swap move at each iteration. To overcome this problem, we use a delete-add compound move that first performs the best delete move from the current solution S_c to generate a partial solution S_c^r and then performs the best add move on S_c^r to reach a new feasible solution S_c, where both the chosen delete and add moves should satisfy that the resulting S_c^r and S_c are not in the tabu lists (see below). Eqs. (4) and (5) give the definition of the best delete move and the best add move, respectively.

$$S_c^r = \arg\min\{f(s_c^r) : s_c^r = S_c \backslash \{i\}, i \in S_c, s_c^r \notin TabuListDLT\} \quad (4)$$

$$S_c = \arg\min\{f(s_c) : s_c = S_c^r \cup \{i\}, i \in J \backslash S_c^r, s_c \notin TabuListADD\} \quad (5)$$

where each partial solution s_c^r is obtained by deleting an element $i \in S_c$ from the solution S_c, and each solution s_c is obtained by adding an element $i \in J \backslash S_c^r$ to the chosen partial solution S_c^r. The best delete-add compound move can be considered as an estimation of the best swap move in terms of the objective function value, which is also used under the name of reduced local search [17] without considering the tabu status of moves.

Another important issue is to design a suitable tabu strategy along with a fast method to determine whether a neighbor solution has a tabu status (i.e., excluded for consideration). For the delete-add compound move, we design two tabu lists where $TabuListDLT$ is used for solutions obtained by performing delete moves and $TabuListADD$ for solutions obtained by performing add moves. A traditional attribute-based tabu strategy needs to set a tabu tenure (for each performed move) which is a parameter whose tuning typically impacts the search performance [20]. An alternative strategy is the solution-based tabu list, which eliminates the need of tuning the tabu tenure [24, 25, 26, 27]. Still recording all visited solutions during the search in a tabu list will require too much memory. Instead of saving each visited solution, we save its hash value. In this way, to verify if a neighbor solution is an eligible candidate solution, it suffices to check if the hash value of this solution is in the tabu list. To map a solution to a hash value, we use a pseudorandom function SipHash-2-4[28],

which is designed to protect against hash collision attacks, while remaining simple to use and fast. On a 64-bit system, the function $hash()$ returns an integer ranging from -2^{63} to $2^{63}-1$, which is large enough for our purpose.

Algorithm 3 shows the general scheme of our solution-based tabu search procedure. Generally speaking, it iteratively moves from one solution to an improved neighbor solution, until the best solution S_l is not improved during the specified d number of consecutive iterations (d is called tabu search depth). For each iteration, it first performs the best delete move with respect to the current solution S_c, subject to the requirement that the hash value of the resulting partial solution S_c^r is not in the tabu list $TabuListDLT$. Meanwhile, the hash value of S_c^r is joined in $TabuListDLT$. Then it performs the best add move with respect to the just obtained partial solution S_c^r, subject to the requirement that the hash value of the resulting solution S_c is not in the tabu list $TabuListADD$. Meanwhile, the hash value of S_c is joined in $TabuListADD$. Finally, as long as an improved solution is found, the best solution S_l is updated.

Algorithm 3 General scheme of STS

```
Input: initial solution S_i, tabu search depth d, tabu lists TabuListDLT and TabuListADD
Output: local optimum S_l
Set S_c = S_l = S_i, n = 0
while n < d do
    Perform the best delete move from S_c to generate the partial solution S_c^r, subject to the requirement that S_c^r is not forbidden by tabu list TabuListDLT
    TabuListDLT = TabuListDLT ∪ {hash(S_c^r)}
    Perform the best add move on S_c^r to generate the feasible solution S_c, subject to the requirement that S_c is not forbidden by tabu list TabuListADD
    TabuListADD = TabuListADD ∪ {hash(S_c)}
    if f(S_c) > f(S_l) then
        S_l = S_c, n = 0
    else
        n = n + 1
    end if
end while
```

4. Preliminary experiments

4.1. Effectiveness of the delete-add compound move

In order to determine whether the objective value obtained after performing the best delete-add compound move is a good estimate of the true objective value obtained after performing the best swap move, we carry out the following experiment. For each of the 144 benchmark instances presented in Section 5, we first randomly generate 1000 initial solutions. For any initial solution, the objective function value of the best solution in N_1 obtained by the swap move is f_{opt}. The estimate of f_{opt} obtained by the delete-add compound move is f_{est}.

If $f_{est} = f_{opt}$, $\#hit = 1$; otherwise, $\#hit = 0$. We summarize $\#hit$ of the 1000 runs and compute *hit rate* $= \sum_{i=1}^{1000} \#hits/1000$.

The delete-add compound move we use is based on a greedy strategy where the best delete move and the best add move are performed. An alternative compound move also based on the greedy strategy is to first perform the best add move and then perform the best delete move. We call the add-delete compound move as estimation-G1 and the delete-add compound move as estimation-G2. In addition to these two compound moves, we also perform experiments for the third compound move that uses a random add move and the best delete move (called estimation-R1) and the fourth compound move that uses a random delete move and the best add move (called estimation-R2).

Figure 1 shows the hit rates of the 144 instances by applying the four compound moves, where the x-axis represents the sorted index of the 144 instances in terms of an non-decreasing order of the estimation-G1 hit rates and the y-axis represents the hit rate of each instance. We observe that estimation-G2 performs the best by reaching a hit rate of 100% for each of the tested 144 instances and estimation-G1 is slightly worse than estimation-G2. On the other hand, the hit rates of estimation-R1 and estimation-R2 are much worse than the two compound moves based on the greedy strategy. To conclude, this experiment demonstrates that the delete-add compound move used in our algorithm is a good approximation of the typical time-consuming swap move in terms of the objective function value.

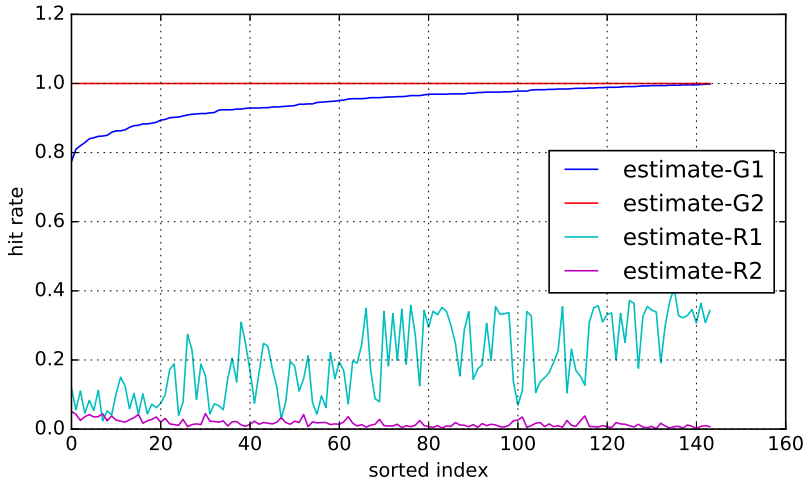

Figure 1: Hit rates of four compound moves

4.2. Impact of using two tabu lists and the delete-add compound move

Our parallel iterative solution-based tabu search algorithm employs the delete-add compound move and two tabu lists as two important search ingredients. In order to verify their effectiveness, we generate the following PISTS variants.

1. PISTS-S1: use the 'add-delete' compound move and a single tabu list,
2. PISTS-D1: use the 'add-delete' compound move and two tabu lists,
3. PISTS-S2: use the 'delete-add' compound move and a single tabu list,
4. PISTS-D2: use the 'delete-add' compound move and two tabu lists.

With the strategy of two tabu lists (PISTS-D1 and PISTS-D2), the resulting partial solution after the first move and the feasible solution after the second move enter the tabu lists each time a compound move is performed, which has the merit of exploring a wider area within the given time limit by strictly preventing solutions from being revisited. With the strategy of a single tabu list (PISTS-S1 and PISTS-S2), we need to consider removing the tabu list used for the partial solutions after the first move or removing the tabu list used for the feasible solutions after the second move. If we remove the partial solutions obtained by performing the first move from the tabu list, i.e., the partial solutions could be revisited, then a sub-optimal feasible solution will be obtained after performing the second move (the optimal feasible solution has entered the tabu list already), which does not comply with the objective. If we remove the feasible solutions obtained by performing the second move from the tabu list, i.e., the feasible solutions could be revisited, then a sub-optimal partial solution will be obtained after performing the first move (the optimal partial solution has entered the tabu list already), which stimulates a deeper exploitation around the revisited feasible solutions. Hence, the single tabu list strategy here means keeping the tabu list for the partial solutions resulted from performing the first move.

To perform this experiment, we run the four variant algorithms on the 144 benchmark instances. We set the time limit for global search T_g to be 3600 seconds, N_k iteration cutoff h to be 1, tabu search depth d to be 10 and the number of processes n_{proc} to be 12. For each PISTS variant, we summarize the number of hits $\#hits$ and the number of improvement $\#Imp$ over small, medium, large and all the instances, respectively. We set $hit = 1$ if the best objective value of solving an instance reaches or surpasses the best known results BKR and $hit = 0$ otherwise. Similarly, we set $Imp = 1$ if the best objective value of solving an instance surpasses the best known results BKR and $Imp = 0$ otherwise. The experimental results of comparing the four PISTS variants are summarized in Table 1, where the best results are marked in bold.

From Table 1, we observe that PISTS-D2 performs the best by matching or surpassing the best known results for 143 instances and discovering the improved results for 7 instances. This discloses the important roles of the two tabu lists and the delete-add compound move to the performance of our algorithm.

Table 1: Comparisons among different PISTS variants

Indicator	Algorithm	Small	Medium	Large	All
#Hit-BKR	PISTS-S1	**48**	47	43	138
	PISTS-S2	**48**	47	44	139
	PISTS-D1	**48**	**48**	45	141
	PISTS-D2	**48**	**48**	**47**	**143**
#New-BKR	PISTS-S1	**0**	**1**	3	4
	PISTS-S2	**0**	0	4	4
	PISTS-D1	**0**	**1**	5	6
	PISTS-D2	**0**	**1**	**6**	**7**

4.3. Parameters analysis of the PISTS algorithm

The N_k iteration cutoff and tabu search depth are two important parameters in our PISTS algorithm. To understand the impact of these parameters, we perform experiments by varying N_k iteration cutoff h in $\{1, 2, 3, 4, 5, 6\}$ and tabu search depth d in $\{10, 20, 30, 40, 50, 60\}$. For each pair (h, d), we perform a single run for all the 144 instances and summarize the ratio of instances that matches the best objective values. The results no less than 140/144 are marked in bold. Table 2 indicates that smaller h generally leads to better results and the setting ($h = 1$, $d = 10$) matches the best objective values for 143 out of 144 instances. This further confirms the rationality of the default parameter settings used in our algorithm.

Table 2: Parameters analysis of the PISTS algorithm

$h\backslash d$	10	20	30	40	50	60
1	**143/144**	**142/144**	**141/144**	139/144	**140/144**	**140/144**
2	**141/144**	**140/144**	**141/144**	138/144	135/144	134/144
3	137/144	138/144	137/144	135/144	135/144	133/144
4	136/144	136/144	135/144	134/144	132/144	130/144
5	133/144	132/144	132/144	130/144	128/144	128/144
6	132/144	130/144	129/144	130/144	127/144	126/144

5. Experimental results and comparisons with state-of-the-art algorithms

The dataset[2] we use contains 144 instances commonly used in the literature [4, 16, 17, 18] where the number of nodes $n = |I| + |J|$ ranges from 400 to 900. These instances are divided into three classes according to the number

[2]http://grafo.etsii.urjc.es/optsicom/opm/opm-files/OpM_LIB_2016.zip

of facilities p (small instances $p = n/16$, medium instances $p = n/8$ and large instances $p = n/4$).

Our PISTS algorithm is written in Python and tested on an AMD Opteron 4184 CPU with 2.8GHz and 32GB RAM. To determine the stopping condition of our algorithm, we refer to the time limit of 600 seconds used by basic variable neighborhood search (VNS) [18], the time limit of 3600 seconds used by branch-and-cut (B&C) [15] and exploring tabu search (XTS) [15], as well as the maximum number of $20p$ iterations used by parallel variable neighborhood search (PVNS). Given that our algorithm and the reference algorithms are executed on different machines, we normalize the computational times by referring to the CPU scores of the Passmark performance test [3]. Using the Intel Core i5 660 CPU as the reference point, the normalized time limit of 3600 seconds on our machine should be $2261/2872 * 3600 = 2834$ seconds. Using the Intel Xeon E7 4820 CPU as the reference point, the normalized time limit of 600 seconds on our machine should be $20212/2872 * 600 = 4222$ seconds. As soon as the running time of performing $20p$ iterations comes or the running time of 2834 seconds is elapsed, we terminate the PISTS run.

5.1. Experimental results of our PISTS algorithm

Tables 3 to 5 present experimental results of the PISTS algorithm. The second column BKR shows the best known results from the state-of-the-art algorithms [17, 18]. Columns 3 to 6 report the best objective values f_{best}, the average objective values f_{avg}, the standard deviation f_{std} and the number of runs reaching BKR over 10 runs $\#hits$. In addition, we also report the average time t_{avg} for a run to reach the best objective value and the standard deviation t_{std} of the computation time over 10 runs. The best objective values of our PISTS algorithm are marked in bold if the best known results are improved.

From Tables 3 to 5, we first observe that our PISTS algorithm is able to find new improved best results for 7 instances and match the best known results for all the other instances. Moreover, the standard deviation is much smaller when compared with the average objective values, indicating that the performance of our PISTS algorithm is quite robust in terms of the objective values. For small and medium instances, the PISTS algorithm is able to reach the best known results in each run. For 43 out of 48 large instances, the PISTS algorithm is still robust by matching or surpassing the best known results in each run. In terms of the computation time, we find that the average time to reach the best objective value generally increases as the number of p grows. Unlike the robust average objective values, the standard deviation of the computation time is relatively large compared with the average computation time.

In summary, the experimental results disclose that our proposed PISTS algorithm is very effective in attaining high quality solutions.

[3] http://www.cpubenchmark.net/

Table 3: Results of our PISTS algorithm for small instances

Instance	BKR	f_{best}	f_{avg}	f_{std}	#$hits$	t_{avg}	t_{std}
pmed17-p25.A	7317	7317	7317.0	0.0	10	1.3	0.4
pmed18-p25.A	7432	7432	7432.0	0.0	10	1.4	0.8
pmed19-p25.A	7020	7020	7020.0	0.0	10	1.2	0.4
pmed20-p25.A	7648	7648	7648.0	0.0	10	1.0	0.2
pmed21-p31.A	7304	7304	7304.0	0.0	10	1.6	0.8
pmed22-p31.A	7900	7900	7900.0	0.0	10	1.7	0.9
pmed23-p31.A	7841	7841	7841.0	0.0	10	1.4	0.9
pmed24-p31.A	7425	7425	7425.0	0.0	10	1.1	1.0
pmed25-p31.A	7552	7552	7552.0	0.0	10	1.7	0.9
pmed26-p37.A	8112	8112	8112.0	0.0	10	2.2	1.4
pmed27-p37.A	7556	7556	7556.0	0.0	10	2.4	1.7
pmed28-p37.A	7366	7366	7366.0	0.0	10	2.2	0.5
pmed29-p37.A	7404	7404	7404.0	0.0	10	2.3	1.2
pmed30-p37.A	7704	7704	7704.0	0.0	10	2.8	1.5
pmed31-p43.A	7424	7424	7424.0	0.0	10	3.9	1.5
pmed32-p43.A	7794	7794	7794.0	0.0	10	4.6	2.3
pmed33-p43.A	7598	7598	7598.0	0.0	10	4.5	2.2
pmed34-p43.A	7725	7725	7725.0	0.0	10	3.2	1.0
pmed35-p50.A	7155	7155	7155.0	0.0	10	3.1	2.1
pmed36-p50.A	8179	8179	8179.0	0.0	10	4.3	4.3
pmed37-p50.A	7830	7830	7830.0	0.0	10	4.5	2.7
pmed38-p56.A	7432	7432	7432.0	0.0	10	9.5	4.9
pmed39-p56.A	7712	7712	7712.0	0.0	10	8.0	2.9
pmed40-p56.A	8211	8211	8211.0	0.0	10	11.1	5.3
pmed17-p25.B	6905	6905	6905.0	0.0	10	2.1	1.4
pmed18-p25.B	7662	7662	7662.0	0.0	10	1.0	0.4
pmed19-p25.B	6816	6816	6816.0	0.0	10	1.3	0.3
pmed20-p25.B	7349	7349	7349.0	0.0	10	1.2	0.9
pmed21-p31.B	7331	7331	7331.0	0.0	10	1.1	0.8
pmed22-p31.B	7695	7695	7695.0	0.0	10	1.9	0.5
pmed23-p31.B	7137	7137	7137.0	0.0	10	2.3	1.5
pmed24-p31.B	7190	7190	7190.0	0.0	10	2.5	0.7
pmed25-p31.B	7552	7552	7552.0	0.0	10	1.5	0.5
pmed26-p37.B	7643	7643	7643.0	0.0	10	2.7	2.2
pmed27-p37.B	7448	7448	7448.0	0.0	10	2.3	1.0
pmed28-p37.B	7388	7388	7388.0	0.0	10	2.9	1.8
pmed29-p37.B	7529	7529	7529.0	0.0	10	2.0	1.0
pmed30-p37.B	8048	8048	8048.0	0.0	10	2.9	1.2
pmed31-p43.B	7320	7320	7320.0	0.0	10	3.7	2.7
pmed32-p43.B	7899	7899	7899.0	0.0	10	3.2	2.0
pmed33-p43.B	7611	7611	7611.0	0.0	10	3.1	1.9
pmed34-p43.B	7514	7514	7514.0	0.0	10	3.4	1.3
pmed35-p50.B	7570	7570	7570.0	0.0	10	4.7	3.4
pmed36-p50.B	8144	8144	8144.0	0.0	10	3.4	2.4
pmed37-p50.B	8379	8379	8379.0	0.0	10	4.8	1.3
pmed38-p56.B	7535	7535	7535.0	0.0	10	5.6	2.2
pmed39-p56.B	7625	7625	7625.0	0.0	10	11.9	5.8
pmed40-p56.B	8022	8022	8022.0	0.0	10	13.2	8.0

Table 4: Results of our PISTS algorithm for medium instances

Instance	BKR	f_{best}	f_{avg}	f_{std}	#$hits$	t_{avg}	t_{std}
pmed17-p50.A	5411	5411	5411.0	0.0	10	7.4	2.5
pmed18-p50.A	5746	5746	5746.0	0.0	10	3.8	2.2
pmed19-p50.A	5387	5387	5387.0	0.0	10	4.4	2.5
pmed20-p50.A	5872	5872	5872.0	0.0	10	3.5	2.2
pmed21-p62.A	5784	5784	5784.0	0.0	10	8.0	4.6
pmed22-p62.A	5995	5995	5995.0	0.0	10	5.9	2.5
pmed23-p62.A	5785	5785	5785.0	0.0	10	9.8	6.9
pmed24-p62.A	5528	5528	5528.0	0.0	10	6.1	4.0
pmed25-p62.A	5767	5767	5767.0	0.0	10	8.8	5.8
pmed26-p75.A	5789	5789	5789.0	0.0	10	18.9	12.9
pmed27-p75.A	5668	5668	5668.0	0.0	10	8.4	5.2
pmed28-p75.A	5681	5681	5681.0	0.0	10	8.1	5.9
pmed29-p75.A	5880	5880	5880.0	0.0	10	12.7	6.6
pmed30-p75.A	6189	6189	6189.0	0.0	10	18.3	10.8
pmed31-p87.A	5905	5905	5905.0	0.0	10	10.6	9.7
pmed32-p87.A	5925	5925	5925.0	0.0	10	33.5	17.5
pmed33-p87.A	5793	5793	5793.0	0.0	10	27.3	19.7
pmed34-p87.A	5849	5849	5849.0	0.0	10	20.8	8.6
pmed35-p100.A	5845	5845	5845.0	0.0	10	15.0	6.3
pmed36-p100.A	6461	6461	6461.0	0.0	10	19.6	10.0
pmed37-p100.A	6203	6203	6203.0	0.0	10	239.4	127.6
pmed38-p112.A	5915	5915	5915.0	0.0	10	240.3	128.5
pmed39-p112.A	5935	5935	5935.0	0.0	10	42.0	33.1
pmed40-p112.A	6272	6272	6272.0	0.0	10	78.7	52.1
pmed17-p50.B	5563	5563	5563.0	0.0	10	2.7	2.3
pmed18-p50.B	5852	5852	5852.0	0.0	10	3.1	2.5
pmed19-p50.B	5423	5423	5423.0	0.0	10	4.7	2.4
pmed20-p50.B	5665	5665	5665.0	0.0	10	4.7	1.4
pmed21-p62.B	5870	5870	5870.0	0.0	10	5.5	3.4
pmed22-p62.B	6259	6259	6259.0	0.0	10	9.6	7.9
pmed23-p62.B	5724	5724	5724.0	0.0	10	6.9	2.7
pmed24-p62.B	5752	5752	5752.0	0.0	10	25.3	15.0
pmed25-p62.B	5692	5692	5692.0	0.0	10	13.0	11.8
pmed26-p75.B	5923	5923	5923.0	0.0	10	8.9	5.2
pmed27-p75.B	5844	5844	5844.0	0.0	10	15.7	12.2
pmed28-p75.B	5642	5642	5642.0	0.0	10	42.4	31.9
pmed29-p75.B	5709	5709	5709.0	0.0	10	9.7	5.9
pmed30-p75.B	6041	6041	6041.0	0.0	10	10.6	7.2
pmed31-p87.B	5621	5621	5621.0	0.0	10	10.4	5.7
pmed32-p87.B	5852	5852	5852.0	0.0	10	137.5	132.5
pmed33-p87.B	5840	5840	5840.0	0.0	10	26.1	17.5
pmed34-p87.B	5857	5857	5857.0	0.0	10	19.8	16.3
pmed35-p100.B	5639	5639	5639.0	0.0	10	57.1	41.8
pmed36-p100.B	6219	6219	6219.0	0.0	10	138.1	114.7
pmed37-p100.B	6211	**6212**	6211.8	0.6	9	24.5	14.9
pmed38-p112.B	5949	5949	5949.0	0.0	10	54.0	36.2
pmed39-p112.B	6198	6198	6198.0	0.0	10	59.3	35.3
pmed40-p112.B	6200	6200	6200.0	0.0	10	176.0	101.8

Table 5: Results of our PISTS algorithm for large instances

Instance	BKR	f_{best}	f_{avg}	f_{std}	#$hits$	t_{avg}	t_{std}
pmed17-p100.A	4054	4054	4054.0	0.0	10	10.9	5.8
pmed18-p100.A	4220	4220	4220.0	0.0	10	62.5	31.1
pmed19-p100.A	4033	4033	4033.0	0.0	10	8.4	1.8
pmed20-p100.A	4063	4063	4063.0	0.0	10	5.1	2.8
pmed21-p125.A	4155	4155	4155.0	0.0	10	85.9	55.5
pmed22-p125.A	4358	4358	4358.0	0.0	10	196.6	119.3
pmed23-p125.A	4114	4114	4114.0	0.0	10	5.2	3.8
pmed24-p125.A	4091	4091	4091.0	0.0	10	28.0	21.9
pmed25-p125.A	4155	4155	4155.0	0.0	10	10.5	4.7
pmed26-p150.A	4341	4341	4341.0	0.0	10	93.4	27.9
pmed27-p150.A	4062	4062	4062.0	0.0	10	52.6	21.5
pmed28-p150.A	4099	4099	4099.0	0.0	10	147.2	91.3
pmed29-p150.A	4141	4141	4141.0	0.0	10	100.2	45.2
pmed30-p150.A	4385	4385	4385.0	0.0	10	27.0	8.5
pmed31-p175.A	4135	**4136**	4135.2	0.4	10	204.0	150.1
pmed32-p175.A	4242	4242	4242.0	0.0	10	229.6	136.1
pmed33-p175.A	4105	4105	4104.4	0.9	7	272.2	136.9
pmed34-p175.A	4287	4287	4287.0	0.0	10	214.2	98.7
pmed35-p200.A	4007	4007	4005.4	2.3	6	183.4	84.6
pmed36-p200.A	4319	4319	4318.4	0.9	7	342.7	159.0
pmed37-p200.A	4593	4593	4593.0	0.0	10	243.2	99.2
pmed38-p225.A	4428	4428	4428.0	0.0	10	260.7	68.9
pmed39-p225.A	4369	4369	4366.8	2.3	4	387.3	152.8
pmed40-p225.A	4571	**4572**	4571.7	0.5	10	259.0	130.5
pmed17-p100.B	3992	3992	3992.0	0.0	10	12.3	7.5
pmed18-p100.B	4122	4122	4122.0	0.0	10	66.9	18.5
pmed19-p100.B	4016	4016	4016.0	0.0	10	2.1	1.5
pmed20-p100.B	4067	4067	4067.0	0.0	10	12.3	8.3
pmed21-p125.B	4033	4033	4033.0	0.0	10	26.1	13.3
pmed22-p125.B	4338	4338	4338.0	0.0	10	34.1	11.8
pmed23-p125.B	4095	4095	4095.0	0.0	10	30.4	26.2
pmed24-p125.B	4072	4072	4072.0	0.0	10	115.3	91.9
pmed25-p125.B	4233	4233	4233.0	0.0	10	22.8	13.6
pmed26-p150.B	4173	4173	4173.0	0.0	10	75.6	34.1
pmed27-p150.B	4144	4144	4144.0	0.0	10	9.1	2.9
pmed28-p150.B	4069	4069	4069.0	0.0	10	248.8	166.2
pmed29-p150.B	4157	4157	4157.0	0.0	10	118.0	35.6
pmed30-p150.B	4313	4313	4313.0	0.0	10	25.0	14.3
pmed31-p175.B	4138	4138	4138.0	0.0	10	11.5	5.5
pmed32-p175.B	4244	**4247**	4244.3	0.9	10	134.9	107.3
pmed33-p175.B	4156	4156	4155.4	1.2	8	163.4	63.3
pmed34-p175.B	4270	4270	4270.0	0.0	10	162.5	42.0
pmed35-p200.B	4109	4109	4109.0	0.0	10	105.7	48.5
pmed36-p200.B	4319	**4321**	4320.4	0.9	10	320.7	191.9
pmed37-p200.B	4609	4609	4609.0	0.0	10	160.7	48.8
pmed38-p225.B	4446	4446	4446.0	0.0	10	26.2	17.2
pmed39-p225.B	4266	**4268**	4267.3	0.6	10	288.3	75.6
pmed40-p225.B	4525	**4532**	4530.6	2.8	10	52.0	15.4

5.2. Comparisons with state-of-the-art algorithms

In this section, we compare our PISTS algorithm with the best performing algorithms in the literature, including greedy randomized adaptive search procedure (GRASP) [4], branch-and-cut (B&C) [15], exploring tabu search (XTS) [15], parallel variable neighborhood search (PVNS) [17] and basic variable neighborhood search (VNS) [18]. For each algorithm, we summarize in Table 6 the

average objective values *Avg.Value*, average computation time *Avg.Time* and the number of hits #*hits* over small, medium, large and all the instances, respectively. The results *Avg.Value* and *Avg.Time* of GRASP, B&C, XTS and PVNS are extracted from [17], which have been obtained on an Intel Core i5 660 CPU with 3.3GHz and 8GB RAM. The results *Avg.Value* and *Avg.Time* of VNS are extracted from [18], which have been obtained on an Intel Xeon E7 4820 CPU with 2.0GHz and 16GB RAM. According to the CPU scores of the Passmark performance test, the running time *Avg.Time* of PISTS and VNS are normalized by using the Intel Core i5 660 CPU as the reference point. Since the best known results BKR used in this paper are better than those used in [17, 18], we update the results #*hits* of each reference algorithm based on the reported detailed results.

From Table 6, we first observe that for the 48 small instances, PISTS performs as well as GRASP, PVNS and VNS by reaching the best known results for each instance and better than B&C and XTS. For the 48 medium instances, PISTS performs best by reaching the best known results for each instance, 3 more instances than VNS, 8 more instances than PVNS, 13 more instances than GRASP, 42 more instances than B&C and XTS. In terms of the average objective values, PISTS performs as well as VNS and better than the other reference algorithms. For the 48 large instances, PISTS performs best by reaching the best known results for each instance, 8 more instances than VNS, 3 more instances than PVNS, 39 more instances than GRASP, 47 more instances than B&C and XTS. Moreover, the average objective value found by PISTS is slightly better than PVNS and VNS and much better than GRASP, B&C and XTS. In terms of the computational time, PISTS performs slightly worse than PVNS and much better than the other algorithms.

Furthermore, we performed the one-sided sign test for the best values of our algorithm and the best performing PVNS and VNS algorithms. The one-sided sign test is a non-parametric test, which assumes the samples are paired and does not require the distribution of the objective values as well as the symmetry of difference. The results indicate that our PISTS algorithm is significantly better than both PVNS and VNS by obtaining the p-values of 0.0059 and 0.0020, respectively.

To conclude, this comparison demonstrates the effectiveness of our proposed PISTS algorithm in terms of both solution quality and computing time.

Table 6: Comparisons with the best performing algorithms in the literatures

Indicator	Algorithm	Small	Medium	Large	All
Avg.BKR		7582.4	5856.9	4213.2	5884.1
Avg.Value	B&C	7452.0	5784.8	4090.3	5775.7
	XTS	7463.0	5814.5	4169.8	5815.8
	GRASP	**7582.4**	5855.5	4205.0	5881.0
	PVNS	**7582.4**	5856.6	4213.1	5884.0
	VNS	**7582.4**	**5856.9**	4212.8	5884.0
	PISTS	**7582.4**	**5856.9**	**4213.3**	**5884.2**
Avg.Time	B&C	1800.0	6915.7	7301.1	5338.9
	XTS	272.4	397.2	331.7	333.8
	GRASP	247.7	406.0	642.4	432.0
	PVNS	8.4	**28.6**	**108.9**	**48.7**
	VNS	23.1	105.6	158.2	95.6
	PISTS	**3.5**	35.8	117.8	52.3
#*Hit-BKR*	B&C	9	6	1	16
	XTS	16	6	1	23
	GRASP	**48**	35	9	92
	PVNS	**48**	40	45	133
	VNS	**48**	45	40	133
	PISTS	**48**	**48**	**48**	**144**

6. Conclusion

In this paper, we proposed a parallel iterative solution-based tabu search algorithm for solving the obnoxious p-median problem. The proposed algorithm is characterized by an effective delete-add compound move to approximate the time-consuming swap move, a solution-based tabu strategy using two tabu lists and a hash function to quickly determine if a solution is previously visited, a randomized perturbation scheme, and a parallel strategy of leveraging multiple processors on a given machine. Extensive experiments on 144 benchmark instances indicate that our algorithm is able to find new lower bounds for 7 instances and match the best known results for all the other instances. Comparisons with state-of-the-art algorithms show the competitiveness of the proposed algorithm in attaining high-quality solutions.

For future research, several directions could be followed. First, the proposed algorithm continuously switches between neighborhoods by performing random moves to achieve search diversification. Combined random and directed moves along with an adaptive selection scheme is worthy of investigation. Second, our findings invite further studies to investigate the benefits of using the compound move combined with the two tabu lists strategy for other cardinality constrained binary optimization problems.

Acknowledgments

We are thankful to the referees for their insightful remarks which improved the paper. This work was partially supported by the National Natural Science Foundation of China (No. 71971172), the National Social Science Fund of China (No. 19BGL093) and the Fundamental Research Funds for the Central Universities (No. 3102019JC03).

References

[1] R. L. Church, R. S. Garfinkel, Locating an obnoxious facility on a network, Transportation Science 12 (2) (1978) 107–118.

[2] R. L. Francis, J. A. White, L. F. McGinnis, Facility layout and location: an analytical approach, Vol. 31, Prentice-Hall Englewood Cliffs, New Jersey, 1974.

[3] P. A. Brown, D. F. Gibson, A quantified model for facility site selection-application to a multiplant location problem, AIIE Transactions 4 (1) (1972) 1–10.

[4] J. M. Colmenar, P. Greistorfer, R. Martí, A. Duarte, Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem, European Journal of Operational Research 252 (2) (2016) 432–442.

[5] J. Current, C. Weber, Application of facility location modeling constructs to vendor selection problems, European Journal of Operational Research 76 (3) (1994) 387–392.

[6] P. Cappanera, A survey on obnoxious facility location problems, Tech. Rep. TR-99-11, Università di Pisa, Dipartimento di Informatica, Pisa, Italy (April 1999).

[7] E. Erkut, S. R. Moran, Locating obnoxious facilities in the public sector: An application of the analytic hierarchy process to municipal landfill siting decisions, Socio-economic Planning Sciences 25 (2) (1991) 89–102.

[8] R. Z. Farahani, M. Hekmatfar, Facility location: concepts, models, algorithms and case studies, Springer, Heidelberg, 2009.

[9] M. J. Kaiser, T. L. Morin, Center points, equilibrium positions, and the obnoxious location problem, Journal of Regional Science 33 (2) (1993) 237–249.

[10] J. E. Beasley, A note on solving large p-median problems, European Journal of Operational Research 21 (2) (1985) 270–273.

[11] Z. Drezner, G. O. Wesolowsky, Obnoxious facility location in the interior of a planar network, Journal of Regional Science 35 (4) (1995) 675–688.

[12] P. Hansen, D. Moon, Dispersing Facilities on a Network, Rutgers University. Rutgers Center for Operations Research, New Jersey, 1988.

[13] M. J. Kuby, Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems, Geographical Analysis 19 (4) (1987) 315–329.

[14] M. Labbé, Location of an obnoxious facility on a network: a voting approach, Networks 20 (2) (1990) 197–207.

[15] P. Belotti, M. Labbé, F. Maffioli, M. M. Ndiaye, A branch-and-cut method for the obnoxious p-median problem, 4OR 5 (4) (2007) 299–314.

[16] G. Lin, J. Guan, A hybrid binary particle swarm optimization for the obnoxious p-median problem, Information Sciences 425 (1) (2018) 1–17.

[17] A. Herrán, J. M. Colmenar, R. Martí, A. Duarte, A parallel variable neighborhood search approach for the obnoxious p-median problem, International Transactions in Operational Research 27 (1) (2018) 336–360.

[18] N. Mladenović, A. Alkandari, J. Pei, R. Todosijević, P. M. Pardalos, Less is more approach: basic variable neighborhood search for the obnoxious p-median problem, International Transactions in Operational Research 27 (1) (2019) 480–493.

[19] A. Tamir, Obnoxious facility location on graphs, SIAM Journal on Discrete Mathematics 4 (4) (1991) 550–567.

[20] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[21] D. L. Woodruff, E. Zemel, Hashing vectors for tabu search, Annals of Operations Research 41 (2) (1993) 123-137.

[22] J. Pei, N. Mladenović, D. Urošević, J. Brimberg, X. B. Liu, Solving the traveling repairman problem with profits: A novel variable neighborhood search approach, Information Sciences 507(2020) 108–123.

[23] J. Pei, J. L. Wei, B. Y. Liao, X. B. Liu, P. M. Pardalos, Two-agent scheduling on bounded parallel-batching machines with an aging effect of job-position-dependent, Annals of Operations Research (2019) DOI: 10.1007/s10479-019-03160-y.

[24] X. Lai, J.-K. Hao, F. Glover, D. Yue, Intensification-driven tabu search for the minimum differential dispersion problem, Knowledge-Based Systems 167 (5) (2019) 68–86.

[25] X. Lai, J.-K. Hao, D. Yue, Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem, European Journal of Operational Research 274 (1) (2019) 35–48.

[26] Y. Wang, Q. Wu, F. Glover, Effective metaheuristic algorithms for the minimum differential dispersion problem, European Journal of Operational Research 258 (3) (2017) 829–843.

[27] Y. Wang, Q. Wu, A. P. Punnen, F. Glover, Adaptive tabu search with strategic oscillation for the bipartite boolean quadratic programming problem with partitioned variables, Information Sciences 450 (29) (2018) 284–300.

[28] J.-P. Aumasson, D. J. Bernstein, Siphash: A fast short-input prf, in: S. Galbraith, M. Nandi (Eds.), Progress in Cryptology - INDOCRYPT 2012, Springer, Berlin, Heidelberg, 2012, pp. 489–508.