Solution-based tabu search for the capacitated dispersion problem

Zhi Lu[a], Anna Martínez-Gavara[b,*], Jin-Kao Hao[c], Xiangjing Lai[d]

[a]*Business School, University of Shanghai for Science & Technology, 516 Jungong Rd., Shanghai 200093, China*
[b]*Dpt. Estadística i Investigació Operativa, Universitat de València, C/Doctor Moliner, 50, 46100 Burjassot, València, Spain*
[c]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[d]*Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

Expert Systems with Applications 223: 119856, 2023, https://doi.org/10.1016/j.eswa.2023.119856

Abstract

Given a weighted graph with a capacity associated to each node (element), the capacitated dispersion problem (CDP) consists in selecting a subset of elements satisfying a capacity constraint, in such a way that the minimum distance among them is maximized. The purpose of this work is to tackle this $\mathcal{NP}$-hard problem, developing a simple, effective, and parameter-free method based on the solution-based tabu search algorithm (`SBTS`). Specifically, we propose a greedy construction heuristic to obtain high-quality initial solutions, and the use of three specific hash functions to identify the tabu status of candidate solutions. Moreover, to enhance the search, we propose the combination of three neighborhoods, including a new one, in which we implemented a constrained swapping strategy. The combination of all these elements results in a very efficient strategy. Extensible computational experiments are performed to get insights into the influences of the algorithmic components of `SBTS`, and to show that our proposal outperforms the state-of-the-art results. Finally, our solution approach is applied to a realistic location problem.

Keywords: Combinatorial optimization, diversity maximization, dispersion, metaheuristics, tabu search.

1. Introduction

The family of dispersion (or diversity) problems arises in a wide variety of practical situations, including facility location, social network analysis, community mining, ecological conservation, and quantum computing. The first discrete models appear in the late eighties with the work of Kuby (1987), and several exact and heuristic methods have been developed to solve them since then (Glover & Laguna, 1998; Duarte & Martí, 2007; Lai et al., 2018). An extended survey of the different mathematical models and methodologies employed for dispersion maximization has been analyzed in Martí et al. (2022). The authors also establish the most appropriate models, and the open problems that are still a challenge.

*Corresponding author.
Email addresses: `zhilusix@gmail.com` (Zhi Lu), `gavara@uv.es` (Anna Martínez-Gavara), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `laixiangjing@gmail.com` (Xiangjing Lai)

Dispersion maximization problems consist in finding a collection of elements from a given set, such that the dissimilarity among them is maximized. Specifically, this dissimilarity (or diversity among the selected elements) is defined differently depending on the practical application example under study, giving rise to several models (Prokopyev et al., 2009; Martí et al., 2022; Martí et al., 2013; Martínez-Gavara et al., 2017). The two most popular diversity models in the literature are the maximum diversity problem (MDP) and the max-min diversity problem (MMDP). In the MDP, the sum of the distances among the selected elements is maximized, while in the MMDP, it is maximized the minimum distance between each pair of elements in the selected set.

The study of Parreño et al. (2021) shows the main features of the solutions obtained with the different diversity models from a geometrical perspective. In particular for MDP and MMDP models, their work reveals that the elements in a solution of the MDP are close to the borders, avoiding the central region; while in the MMDP, the points are equidistant all over the space, and it does not avoid to select points in the central region. The authors conclude that the MMDP is thus recommended for some location problems, such as obnoxious facilities. These models, however, usually neglect the introduction of constraints, and rely only on the selection of a prefixed number of elements, which makes them somehow unrealistic.

This work focuses on a new model based on the MMDP that replaces the cardinality restriction with a capacity constraint. This model, known as the capacitated dispersion problem (CDP), is $\mathcal{NP}$-hard and was first defined by Rosenkrantz et al. (2000) in the context of locating facilities. Specifically, the CDP seeks to maximize the dispersion of the selected elements (sites, in facility location applications) under a capacity constraint, which models a minimum storage capacity requirement.

The mathematical model for the CDP is defined as follows. Given a graph $G = (V, E)$, where V is a set of n nodes (elements) and E is a set of edges, let c_i be the capacity of node $i \in V$, and let d_{ij} be the distance between nodes i and j, $\forall (i, j) \in E$ ($d_{ij} = 0$, if $(i, j) \notin E$). The CDP consists of selecting a subset $M \subseteq V$, such that the sum of the selected nodes capacities is at least a required value B, while maximizing the minimum distance between the pairs of selected elements. The CDP can be formulated with binary variables x_i ($i = 1, \ldots, n$) that take the value 1 if element i is selected, and 0 otherwise. Then, the mathematical programming model (Peiró et al., 2021) is defined as follows:

$$\textbf{Maximize} \quad \min_{i,j \in M} d_{ij} \tag{1}$$

$$\textbf{subject to:} \quad \sum_{i=1}^{n} c_i x_i \geq B \tag{2}$$

$$M = \{i \in V : x_i = 1\} \tag{3}$$

$$x_i \in \{0, 1\}, \ i = 1, 2, \ldots, n; \tag{4}$$

where constraint (2) ensures that the minimum capacity requirement (B) of the selected elements is satisfied. A solution M can be represented by a n-dimensional array $x = (x_1, x_2, \ldots, x_n)$ with $x_i = 1$ if element $i \in M$ and $x_i = 0$ otherwise ($i = 1, 2, \ldots, n$). The quality of any candidate solution M in the search space Ω,

$$\Omega = \{M \subset V, \ 2 \leq |M| \leq n\}, \tag{5}$$

is given by its objective function value $f(M)$ as shown in Equation (1).

Some undesirable facilities applications, such as landfills or sewage plants, nuclear or chemical plants, and hazardous wastes sites, should be dispersed enough to reduce human and environmental harm while satisfying storage capacity limitations and cost constraints. This problem also finds an application in retail franchises, where shops should be placed dispersed with a minimum capacity of service to satisfy the overall demand. More examples can be found in the work of Martínez-Gavara et al. (2021), where the authors introduce other practical and realistic applications that can be easily adapted to this problem.

The importance of the practical applications of the CDP merits further study and motivates this work. As mentioned above, classical models such as MDP or MMDP, consider a prefixed number of facilities without additional constraints; however, in practical problems, the capacity of each facility depends on its location, so the number of facilities is adjusted in real time. The CDP models these characteristics to approach real location problems in a better way than the classical ones. In fact, one important motivation to deal with this $\mathcal{NP}$-hard optimization problem is to solve a realistic location problem proposed in Daskin (2011), and studied later on in Lozano-Osorio et al. (2022), where an international branch company wants to open new offices in U.S. to expand its business. This practical problem is introduced in detail and solved in Section 5.

One of the main goals of this paper is to propose a simple but effective approach to deal with the CDP. In particular, we propose a meta-heuristic based on Tabu Search methodology (Glover & Laguna, 1998), which has been proven to be very efficient in diversity problems (Martí et al., 2022; Duarte & Martí, 2007; Porumbel et al., 2011). The typical implementation to avoid cycling is attribute-based memory; however, in this paper, we consider a less studied methodology, the solution-based tabu search method, where the complete solution is recorded. The use of specific hash functions speeds up the process of identifying the solution, obtaining a competitive meta-heuristic (Lai et al., 2018). Specifically, each global iteration of our method consists in applying a greedy heuristic to construct a high-quality initial solution in a very short computational time, and then the algorithm combines a large neighborhood with the solution-based memory to further improve the best solution found so far.

The remainder of the paper is structured as follows. We first review the related literature and highlight the main contributions of this work in Section 2. Section 3 describes the proposed solution-based tabu search algorithm for the CDP. Section 4 reports the computational evaluations of the proposed `SBTS` algorithm, and Section 5 applies the approach to solve a realistic location problem. Section 6 analyzes key algorithmic ingredients, and Section 7 concludes the work and provides perspectives for future research. Appendix provides detailed instance-by-instance comparison results between `SBTS` and the reference algorithms.

2. Literature review

This section is limited to reviewing the related literature. We first review the literature addressing the CDP, and then the papers that apply the solution-based tabu search methodology in diversity problems.

To the best of our knowledge, there are only three recent studies in the literature for solving the CDP. The first work by Rosenkrantz et al. (2000), proposes a simple greedy heuristic, called `TI`, with a $\mathcal{O}(n^2 \log n)$ running time. This work is mainly theoretical and there are no reported empirical results or experiments, though the authors demonstrate a good performance guarantee of `TI`.

A different approach is developed in Peiró et al. (2021), who proposes a strategic oscillation method, named SO, which combines a greedy randomized adaptive search procedure (GRASP) and a variable neighborhood descent (VND) to tackle the CDP. The authors also present a mathematical model to solve small-sized and medium-sized instances with optimal results. An empirical comparison on 100 benchmark instances shows the superiority of SO compared to the previous heuristic TI, and the mathematical model solved using CPLEX and LocalSolver software.

Following that, Martí et al. (2021b) proposes a new mathematical model and devise a heuristic based on scatter search for the CDP, called SS. Their experimentation show that the new mathematical model solves to optimality many more instances than the previous one, and the SS method outperforms the previous one SO as well.

The three previous papers greatly contributed to better solving the CDP. However, two main limitations can be identified in the existing approaches, which motivate the current work.

- First, these approaches have difficulties in robustly and consistently producing high-quality solutions for large-sized graphs.
- Second, they may require a substantial amount of computation time to reach satisfactory results.

We thus conclude that the research on solving the CDP is still in its early stages, and effective algorithms, especially heuristic methods, need to be developed further to address this challenging problem.

On the other hand, in the review paper of Martí et al. (2022), the authors conclude that Tabu Search (TS) is among the best metaheuristics to solve diversity maximization problems. For this reason, in Section 3, we propose a procedure based on the tabu search methodology for the CDP. In particular, we consider an interesting alternative to the classical tabu search that focuses on solution attributes (ABTS) (Galinier & Hao, 1999; Glover & Laguna, 1998; Lu et al., 2020, 2019; Wang et al., 2014). We propose a solution-based tabu search (SBTS) method (Lai et al., 2019b, 2018; Wang et al., 2017; Wei & Hao, 2021) that records visited solutions (instead of attributes) to avoid search cycling. This variant has received considerably less attention in the literature, and as far as we know, only five works propose a solution-based tabu search methodology to solve a diversity problem. See the timeline diagram in Figure 1. Next, these five recent and interesting papers are briefly described.

In 2017, Wang et al. (2017) presents SBTS and memetic algorithms for the challenging Minimum Differential Dispersion Problem (MinDiff DP), first showing the superiority of the SBTS for tackling the diversity problems. In 2018, Lai et al. (2018) proposes an SBTS method with a parametric swap neighborhood for the Maximum Min-sum Dispersion Problem (MaxMinSum DP). Experiments show that their method outperforms the state-of-the-art methods in both solution quality and computational efficiency.

Following that, in 2019, an intensification-driven tabu search algorithm is developed in Lai et al. (2019a) for the MinDiff DP with the outstanding experimental results. Subsequently, a SBTS method integrated with dynamic neighborhood size and two new hash functions strategies is designed by Lai & Fu (2019) to solve the MaxMinSum DP, and experiments show that they work very well, especially for large-scale instances.

Finally in 2021, Wang et al. (2021) proposes a two-phase intensification tabu search algorithm for the MaxMinSum DP, for which both integrates attribute-based tabu search and solution-based tabu search to refine the local search procedure. The above recent developments

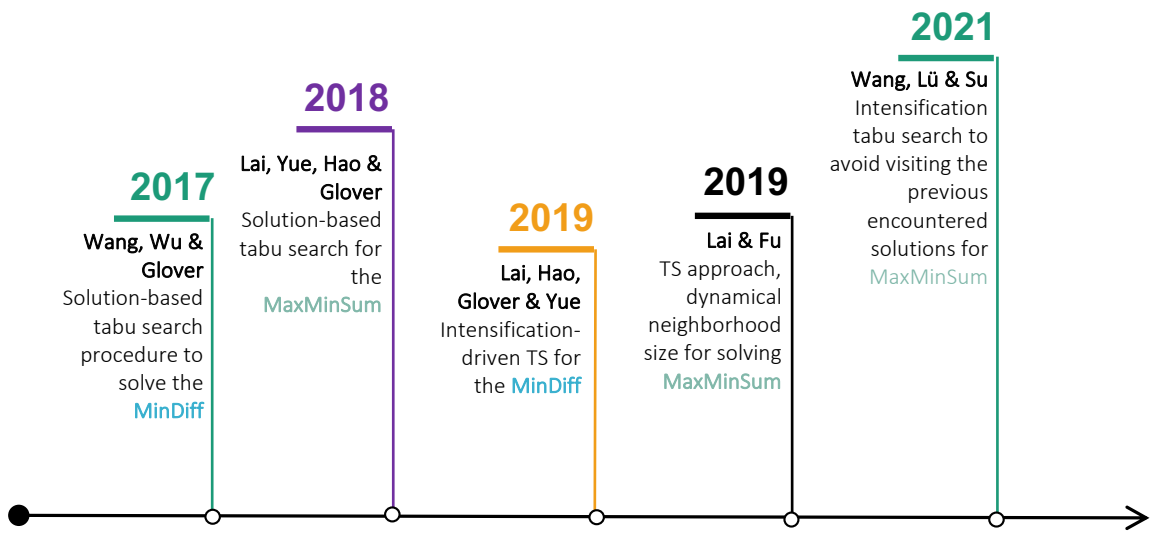


Figure 1: Timeline of the SBTS method in diversity maximization problems.

support the use of the SBTS method to deal with diversity problems, and in particular, with the CDP.

However, as far as we know, this methodology has not yet been tested, and therefore, has not been adapted to max-min or min-max problems. It is well-known that max-min (min-max) problems are a challenge for heuristic search since there exists many solutions with the same objective function value. Part of the difficulty is that changes in the solutions cannot be identified by changes in the objective function. To overcome this issue, (i) we devise a tabu search meta-heuristic which can identify the already visited solutions using three hash functions, and on the other hand, (ii) in the improvement phase a new constrained swap neighborhood is proposed. This neighborhood only involves swap moves between elements in two specifically subsets, which guarantee the improvement of the current objective function value, under feasibility conditions.

We had a twofold goal for this work: to experiment with the implementation of solution-based tabu search on a max-min problem and, in the process, to develop a state-of-the-art procedure for the CDP. The following contributions have allowed the achievement of both goals:

(i) to propose a simple and effective solution-based tabu search algorithm (`SBTS`) for the CDP. Our proposal is simple in design and parameter-free, relieving users of the time-consuming task of parameter calibration;

(ii) to incorporate a hash-based mechanism to the `SBTS` algorithm to quickly identify the tabu status of neighboring solutions in the flat landscape presented by the max-min problem;

(iii) to propose a greedy construction heuristic (`GreedyIS`) to create a high-quality initial solution;

(iv) to propose a new neighborhood, the constrained Swap (CSwap). The union of Add, Drop, and constrained Swaps moves promote a more complete neighborhood exploration

in a given time frame;

(v) to present a streamline calculation strategy ($\mathcal{O}(n)$) to quickly determine the changed of the objective function value;

(vi) to assess our proposal on the 100 benchmark instances commonly used in the literature. Our SBTS algorithm outperforms the state-of-the-art algorithms for most of the benchmark instances. Specifically, SBTS finds the optimal results for 73 out of 80 instances for which it is known, and it improves the best-known results for the remaining 20 instances with unknown optimal solutions.

This work demonstrates the benefit of the solution-based tabu search method for tackling this challenging dispersion (or diversity) problem. The code of the SBTS algorithm will be publicly available, allowing researchers to better solve various practical problems that can be formulated as the CDP and the related dispersion (or diversity) problems.

3. Solution-based tabu search for the capacitated dispersion problem

Many metaheuristics can be used to tackle this hard optimization problem. Techniques based on artificial intelligence, social behavior or bio-inspired have demonstrated to efficiently solve many optimization problems (Glover & Laguna, 1998; Neri & Cotta, 2012; Ruiz & Stützle, 2007; Martí et al., 2006; Wang et al., 2014). As mentioned in Section 2, in Martí et al. (2022), the authors review the literature on diversity maximization problems from an OR perspective and perform an empirical review and comparison of the best and more recently proposed algorithms to solve them. Based on their experimentation, Tabu Search (TS) emerges the best metaheuristic to solve diversity maximization problems, where the simple TS proposed by Porumbel et al. (2011) solves the best one for the MaxMin dispersion problem. It is clear, though, that the use of memory is a key point in the ability of a metaheuristic in search for a global optimum in diversity maximization problems. Therefore, it is quite natural to use TS as the methodology to solve the CDP.

Tabu search (Glover & Laguna, 1998) is a metaheuristic that has been successfully applied in many combinatorial optimization problems, not only in diversity maximization problems. Although most studies use attribute-based adaptive memory for guiding purposes, the recent works of Lai et al. (2019b) and Wei & Hao (2021) highlight the effectiveness of the solution-based tabu search procedure to solve binary optimization problems.

In this work, following the principle of solution-based tabu search presented in Lai et al. (2019b, 2018); Wang et al. (2017); Wei & Hao (2021), we devise an effective solution-based tabu search algorithm (SBTS) for the CDP. This is the first time that this strategy has been adapted to the CDP. The next sections go through the major components of our SBTS algorithm.

3.1. General procedure

In general terms, the SBTS algorithm starts with an initial solution (M) obtained by a greedy heuristic procedure, and then it applies a tabu search strategy that operates as follows. It scans the neighborhood of the current solution and evaluates all possible solutions that can be obtained by one of the following moves: insertion of a new element ($Add(\cdot)$), extraction of an element ($Drop(\cdot)$), or the exchange between a selected element with an non-selected one ($CSwap(\cdot)$). Then, the algorithm performs the move that produces the best new solution M'. As it is customary in the TS methodology, the algorithm always performs the best available

Algorithm 1: Solution-based tabu search for the CDP (SBTS).

Input: Graph $G = (V, E)$, hash tables H_1, H_2, H_3, hash functions h_1, h_2, h_3, length of hash tables l, cutoff time t_{max}
Output: Best solution M^* found during the search

1 **begin**
  /* Initialize the hash tables, Section 3.5 */
2 **for** $i = 0$ *to* $l - 1$ **do**
3 $H_1[i] \leftarrow 0$
4 $H_2[i] \leftarrow 0$
5 $H_3[i] \leftarrow 0$
  /* Initial solution with greedy construction heuristic, Section 3.2 */
6 $M \leftarrow$ GreedyIS() /* Algorithm 2 */
7 $M^* \leftarrow M$ /* M^* records the best solution found so far */
8 **repeat**
  /* Explore the neighborhood of the current solution M */
9 $N(M) = N_{Add} \cup N_{Drop} \cup N_{CSwap}$ /* Section 3.3 */
10 $M' \leftarrow$ best_nontabu_solution($N(M)$)
11 $M \leftarrow M'$ /* Update the current solution M */
  /* Update the best solution found so far */
12 **if** $f(M) > f(M^*)$ **then**
13 $M^* \leftarrow M$
14 $f(M^*) \leftarrow f(M)$
  /* Update the hash tables with M, Section 3.5 */
15 $H_1[h_1(M)] \leftarrow 1$
16 $H_2[h_2(M)] \leftarrow 1$
17 $H_3[h_3(M)] \leftarrow 1$
18 **until** $time() \geq t_{max}$;
19 **return** M^*

move, even if it is a non-improving move. The new solution is declared visited, i.e., tabu in TS terminology, by using the hash functions and hash tables. This mechanism avoids search cycling. The algorithm finishes when a time limit is reached, and the best visited solution is returned as its output.

Algorithm 1 shows the general procedure of our method.To reduce the CPU time of determining the tabu status of a candidate solution, SBTS uses three hash tables associated with three hash functions, respectively, where H_k ($k = 1, 2, 3$) represents the hash tables, h_k ($k = 1, 2, 3$) identifies the corresponding hash functions, and l is the length of the hash tables. These hash functions help the search to avoid previously-visited solutions (Lai & Fu, 2019; Lai et al., 2018; Wei & Hao, 2021).

First, SBTS initializes the hash tables (Alg. 1: lines 2-5, and described in Section 3.5) and generates an initial solution M by a greedy construction heuristic (GreedyIS) (Alg. 1: line 6, detailed in Section 3.2). A greedy construction is preferred rather than a randomized one, since the quality of the initial solution favors the algorithm to convergence faster and better (Laguna et al., 1999; Martínez-Gavara et al., 2021). M^* records the best solution found so far (Alg. 1: line 7).

SBTS then performs moves as it is customary in tabu search, until the given stopping condition is met, which is typically a cutoff time limit (Alg. 1: lines 8-18). At each iteration,

Algorithm 2: Greedy construction heuristic for initial solution (`GreedyIS`).

Input: Graph $G = (V, E)$, total required capacity B
Output: Initial solution M

1 **begin**
2 $M \leftarrow \emptyset$, $c(M) \leftarrow 0$ `/* Initial` M `and the total weight` $c(M)$ `*/`
3 $M \leftarrow M \cup \{v_0\}$ `/* Start with a random seeding node` $v_0 \in V$ `*/`
4 $c(M) \leftarrow c(M) + c_{v_0}$
5 **repeat**
6 Choose v at random from $\hat{V} = \arg\max_{v \in V \setminus M} de(v)$, where $de(v) = \min_{u \in M} d_{uv}$
7 $M \leftarrow M \cup \{v\}$
8 $c(M) \leftarrow c(M) + c_v$
9 **until** $c(M) \geq B$;
10 **return** M

`SBTS` first identifies the best non-tabu neighboring solution M' from the union of N_{Add}, N_{Drop}, and N_{CSwap} neighborhoods (Alg. 1: lines 9-10), and then moves from M to M' (Alg. 1: line 11), see Section 3.3 for more details. The main motivation of this phase of the algorithm is twofold. On the one hand, to diversify the search by iteratively transitioning from the current solution M to a nearby solution M' in its neighborhood ($N(M)$), and on the other hand, to enhance the search by selecting the best admissible candidate among neighboring solutions within the neighborhood. The current solution M is replaced by M' although the best admissible neighboring solution M' is not better than M. The algorithm is then able to explore new regions of the solution space, without getting trapped in a local optima (Wei & Hao, 2021).

Subsequently, the hash tables are updated according to the new solution M (Alg. 1: lines 15-17, described in Section 3.5). This prevents the search from visiting the encountered solution M again. The loop is repeated until the cutoff time (t_{max}) is reached (Alg. 1: line 18). Finally, the best solution M^* found during the search process is returned as the final result (Alg. 1: line 19).

In the following sections, we describe each of the main components of the `SBTS` algorithm (Algorithm 1). Section 3.2 presents the greedy construction heuristic. Then, Section 3.3 explains how the three neighborhoods used in this work are built. In addition, Section 3.4 gives details about the streamline calculations that quickly evaluate the weight of each neighboring solution. Finally, Section 3.5 describes the hash functions and hash tables, which determine the tabu status of a neighboring solution.

3.2. Initial solution with greedy construction heuristic

To generate an initial solution of good quality, the proposed `SBTS` algorithm applies a simple greedy construction heuristic (`GreedyIS`, Algorithm 2). Let $c(M)$ be the total capacity of the current solution M, computed as the sum of the capacities c_i of the elements i in M. M and $c(M)$ are initially set to $\emptyset$ and 0, respectively (line 2). `GreedyIS` begins with a random seeding element v_0 (lines 3-4) and then, in the subsequent steps, the procedure constructs an initial solution by iteratively adding one element at a time to the partial solution. At each iteration, an element v is chosen at random from those with the largest minimum distance between the unselected elements ($v \in V \setminus M$) and the selected ones ($u \in M$). Specifically, 1) `GreedyIS` first calculates the minimum distance $de(v) = \min_{u \in M, v \in V \setminus M} d_{uv}$, for each unselected element

($v \in V \setminus M$) to the selected elements ($u \in M$); and then 2) chooses an unselected element v with the largest $de(v)$ value (ties break at random) to insert into M, and increase $c(M)$ by the capacity of element v, c_v. The two steps are repeated until M meets the total required capacity B (lines 5-9). Since the CDP consists in maximizing the interdistance value within M, the larger a $de(v)$ value, the better a candidate element v. The time complexity of the `GreedyIS` procedure is bounded by $\mathcal{O}(m \times n)$, where m is the number of selected elements ($m = |M|$).

3.3. Neighborhood structures

To effectively explore the search space Ω (5), `SBTS` explores the neighborhood defined by a joint use of three move operators $Add(\cdot)$, $Drop(\cdot)$, and Constrained $Swap$ ($CSwap(\cdot,\cdot)$). Among them, $Add(\cdot)$ and $Drop(\cdot)$ are the basic move operators that have been successfully applied in previous works (Martí et al., 2021b; Peiró et al., 2021) for local improvement, while $CSwap(\cdot,\cdot)$ is first introduced in this work.

Given a solution M, an element $u \in M$, and $v \in V \setminus M$, the basic move operators are briefly described as follows,

- ***Add*(v)** adds an element $v \in V \setminus M$ to the set M to generate a neighboring solution M'. Formally, the neighborhood defined by this operator is given below:

$$N_{Add}(M) = \{M' : M' = M \oplus Add(v),\ v \in V \setminus M\}. \tag{6}$$

- ***Drop*(u)** removes an element $u \in M$ to generate a neighboring solution M', such that the capacity constraint is respected, i.e., $c(M) - c_u \geq B$. Formally, the neighborhood defined by this operator is given below:

$$N_{Drop}(M) = \{M' : M' = M \oplus Drop(u),\ u \in M,\ c(M) - c_u \geq B\}. \tag{7}$$

One observes that $Add(v)$ and $Drop(u)$ operators lead to a neighborhood of size $\mathcal{O}(n-m)$ and $\mathcal{O}(m)$, respectively. $Drop(u)$ can either improve or have no effect on the quality of the current solution, while $Add(v)$ always decreases or has no effect on the objective function value.

The $Swap(u,v)$ operator (Peiró et al., 2021) often exchanges an element $u \in M$ with an element $v \in V \setminus M$ to generate a neighboring solution M', resulting in a neighborhood of size $\mathcal{O}(m \times (n-m))$, which is very large and computationally expensive. To improve on the computational efficiency of `SBTS`, we devise a new constrained swap neighborhood that limits the swap moves to be examined in two specifically identified subsets $X \subseteq M$ and $Y \subseteq V \setminus M$. The induced move operator $CSwap(u,v)$ is defined as follows:

- ***CSwap*(u,v)** limits the swapped elements to two high-quality subsets $u \in X \subseteq M$ and $v \in Y \subseteq V \setminus M$ to generate a neighboring solution M'. Formally, the neighborhood induced by this operator is given below:

$$N_{CSwap}(M) = \{M' : M' = M \oplus CSwap(u,v),\ u \in X, v \in Y\}. \tag{8}$$

The subsets X and Y can be identified by developing the idea introduced in Martí et al. (2021b) to remove or replace those elements in the solution in which the objective function matches its value. For all $u \in M$, let $di(u)$ be the minimum distance between a selected element

$u \in M$ and the rest of the elements in the solution, M, i.e., $di(u) = \min_{x \in M} d_{ux}$. Note that the objective function value $f(M)$ is to maximize the $di(u)$ value. So, it is clear that to improve a current solution M, the element (or elements) $u \in M$ for which $di(u) = f(M)$ must be removed or replaced; this is why they are called critical elements. The subset X then includes all possible elements $u \in M$ for which $di(u) = f(M)$, i.e., $X = \{u \in M : di(u) = f(M)\}$. The set $V \setminus M$ is much larger than the set M, we thus implement an efficient way to identify a good exchange $v \in Y \subseteq V \setminus M$ for the elements $u \in X$. Specifically, we compute for $v \in V \setminus M$, $de(v) = \min_{x \in M} d_{xv}$ as in Section 3.2. Since $f(M)$ consists in maximizing pairwise distances within M, the larger $de(v)$ the better the insertion of element v. We thus collect all possible elements v with the largest (max) and the second largest (secmax) $de(v)$ values to build the subset Y, i.e., $Y = \arg\max_{v \in V \setminus M} de(v) \ \cup \ \arg\text{secmax}_{v \in V \setminus M} de(v)$. The constrained swap neighborhood is more efficient since the subsets X and Y are much smaller than the sets M and $V \setminus M$. In practical terms, the worst-case complexity is $\mathcal{O}(m \times n)$, however, this is not reached for most iterations. The average complexity only requires $\mathcal{O}(1)$ time (as observed in our experiments, $|X| \in [1, 10]$ and $|Y| \in [1, 10]$ in most iterations).

It is well documented in the tabu search literature (Glover & Laguna, 1998) that it is usually better to define a large neighborhood that is constrained or filtered according to problem information, than to consider a small neighborhood that limits the exploration by definition.

The neighborhood explored by `SBTS` covers the above three neighborhoods, i.e., $N(M) = N_{Add} \cup N_{Drop} \cup N_{CSwap}$. At each iteration, `SBTS` scans the current neighborhood $N(M)$ and applies a streamline calculation strategy (Section 3.4) to quickly evaluate each neighboring solution. A best non-tabu candidate solution M' is then selected to replace the current solution M. Once all candidate solutions in $N(M)$ have been prohibited by the tabu list, the best candidate solution M', regardless of its tabu status, is selected to replace the current solution M (aspiration criterion).

3.4. Streamline calculations

To quickly determine the objective function value change and, in order to reduce the computational complexity, we present here the streamline calculation strategy for the CDP. Let M be the current solution and M' be a neighboring solution after applying the $Add(\cdot)$, $Drop(\cdot)$, or $CSwap(\cdot, \cdot)$ move operators to the current solution M. It is worthy noting that the objective function value $f(M)$ is calculated by maximizing the minimum distance between the pairs of selected elements in M. We thus maintain and update the following two n-dimensional arrays:

$$md(v) = \min_{u \in M} \ d_{uv}, \ \forall v \in V. \tag{9}$$

$$mdc(v) = |\{u \in M, d_{uv} = md(v)\}|, \ \forall v \in V. \tag{10}$$

where $md(v)$ records the distance between each element $v \in V$ and the selected elements $u \in M$, and $mdc(v)$ counts the number of neighbors of element v, whose distance to v are equal to $md(v)$. Note that $md(v) = de(v)$ if $v \in V \setminus M$ and, $md(v) = di(v)$ if $v \in M$.

For a given solution M, the objective function value of a neighboring solution M' after an $Add(v)$ move to M can be quickly determined as:

$$f(M') = \begin{cases} f(M) & , \text{ if } md(v) \geq f(M) \\ md(v) & , \text{ if } md(v) < f(M). \end{cases} \tag{11}$$

Similarly, the objective function value of a new solution M' after a $Drop(u)$ move to M can be conveniently calculated as:

$$f(M') = \begin{cases} f(M) & , \text{ if } md(u) > f(M) \vee (md(u) = f(M) \wedge mdc(u) > 1) \\ \text{Recalculate } f(M) & , \text{ if } md(u) = f(M) \wedge mdc(u) = 1. \end{cases} \tag{12}$$

After each add operation (i.e., adding an element v from set $V \setminus M$ to set M), it is necessary to examine all elements $x \in V \setminus \{v\}$ and update the two arrays $md(x)$ and $mdc(x)$ as follows:

Case 1 **if** $d_{xv} < md(x)$, **then** $md(x) = d_{xv} \wedge mdc(x) = 1$

Case 2 **if** $d_{xv} = md(x)$, **then** $mdc(x) = mdc(x) + 1$

Case 3 **if** $d_{xv} > md(x)$, **then** keep $md(x)$ and $mdc(x)$ unchanged

After each drop operation (i.e., dropping an element u from set M to set $V \setminus M$), it is required to examine all elements $x \in V \setminus \{u\}$ and update the two arrays $md(x)$ and $mdc(x)$ as follows:

Case 1 **if** $d_{ux} = md(x) \wedge mdc(x) = 1$, **then** recalculate $md(x)$

Case 2 **if** $d_{ux} = md(x) \wedge mdc(x) > 1$, **then** $mdc(x) = mdc(x) - 1$

Case 3 **if** $d_{ux} > md(x)$, **then** keep $md(x)$ and $mdc(x)$ unchanged

The records for elements u and v do not require any modification. Note that the constrained swap operation does not require the streamline calculation strategy here since its neighborhood size is relatively small and limited by $\mathcal{O}(1)$. The swap can also be separated into the add and drop operations, we may achieve quick updating for the constrained swap operation by employing the above streamline calculation strategy.

We now discuss the time complexity of the streamline calculation strategy. The objective function value $f(M')$ (Equation (11) and Equation (12)) can be quickly calculated in $\mathcal{O}(1)$ time for most cases. However, if $md(u) = f(M) \wedge mdc(u) = 1$ (Equation (12)), recalculating $f(M)$ takes $\mathcal{O}(n)$ time. In most situations, updating $md(x)$ and $mdc(x)$ after the add or drop operations takes $\mathcal{O}(n)$ time but the drop operation (Case 1), recalculating $md(x)$ is required with $\mathcal{O}(m \times n)$ time. In fact, the above recalculations are not required at each iteration, thus the streamline calculation strategy requires a total of $\mathcal{O}(n)$ time.

3.5. Solution-based tabu strategy using hash functions

Tabu search typically uses an attribute-based definition of tabu status to identify moves that cannot be considered during a specific period of time (tabu tenure). To avoid the search from revisiting previously encountered solutions, solution attributes (such as a node, edge, value of a variable, etc.) are recorded in short-term memory (tabu list). Attribute-based tabu search (ABTS) is a general and powerful metaheuristic that has been utilized to tackle many combinatorial optimization problems (Galinier & Hao, 1999; Glover & Laguna, 1998; Lu et al., 2020, 2019; Wang et al., 2014). Recently, solution-based tabu search (SBTS) has emerged as an interesting alternative that avoids search cycling by recording visited solutions rather than attributes, making use of hash functions to determine tabu status of distinct solutions (Lai et al., 2019b, 2018; Wei & Hao, 2021). SBTS has gradually gained some research attention because of its ability to free up tuning tabu tenure; however, the high running time associated with recording complete solutions needs to be addressed.

Generally, given a hash function h and a hash table H of size l, h can be used to map a candidate solution $M' \in \Omega$ to an index of H, i.e., $h : M' \in \Omega \rightarrow \{0, 1, 2, \ldots, l-1\}$, with a binary value of $H[h(M')]$ to identify the tabu status of solution M'. Specifically, $H[h(M')] = 1$ indicates that the solution M' has been visited previously and is classified as tabu (thus M' is

not eligible for consideration at the current iteration unless the aspiration criterion is met), while $H[h(M')] = 0$ indicates that M' has not been visited and is thus eligible for consideration at the current iteration. Our tabu list management strategy employs multiple hash tables and hash functions to greatly reduce the probability of wrong identification of the tabu status. Particularly, we use three hash tables H_k $(k = 1, 2, 3)$ of length l, where each position takes a binary value that contributes to the definition of the tabu status of candidate solutions. The hash tables are all initialized to 0, indicating that no candidate solutions are classified as tabu. Once a candidate solution M' is selected to replace the current solution M, the corresponding positions in the three hash tables will be set to 1, i.e., $H_k[h_k(M')] \leftarrow 1$ $(k = 1, 2, 3)$.

Given a solution $x = \{x_1, x_2, \ldots, x_n\}$ (also called M) where $x_i = 1$ if element $i \in M$ and $x_i = 0$ otherwise $(i = 1, 2, \ldots, n)$, the three hash functions $h_k(M)$ $(k = 1, 2, 3)$ (Lai & Fu, 2019) are defined as follows,

$$h_k(M) = (\sum_{i=1}^{n} w_k(i) \times x_i) \ mod \ l. \tag{13}$$

where $w_k(i) = w_k(i-1) + \beta_k + rand(\beta_k/2)$, $w_k(0) = \beta_k$, β_k $(k = 1, 2, 3)$ is a parameter which takes different values (set to 300, 400, and 500 respectively in this work) for three hash functions (Table 2, Section 4.2), $rand(\beta_k/2)$ takes a random integer value between 0 and $\beta_k/2$, and l is the length of hash tables (empirically set to 10^8 in this work).

For a solution M and its hash function $h_k(M)$ $(k = 1, 2, 3)$, we can quickly calculate the hash function values of its neighboring solution M' resulting from the $Add(v)$ or $Drop(u)$ or $CSwap(u, v)$ move operators as follows,

$$h_k(M') = \begin{cases} h_k(M) + w_k(v) & , \text{ if } v \in V \setminus M \text{ for the add operation} \\ h_k(M) - w_k(u) & , \text{ if } u \in M \text{ for the drop operation} \\ h_k(M) - w_k(u) + w_k(v) & , \text{ if } u \in M, v \in V \setminus M \text{ for the swap operation.} \end{cases} \tag{14}$$

The following is how the hash-based tabu list management strategy works. Each hash function is initially associated with a hash table of length l $(|l| = 10^8)$, and each hash table is initialized to 0. For a solution M, we calculate the three hash function values $h_k(M)$ $(k = 1, 2, 3)$ to identify the index of the three hash tables. The tabu status of a solution M is determined according to the values of the three hash tables $H_k[h_k(M)]$ $(k = 1, 2, 3)$. Specifically, M is classified as a prohibited solution (i.e., already visited) when $H_1[h_1(M)] \wedge H_2[h_2(M)] \wedge H_3[h_3(M)] = 1$. Otherwise, M is determined as an unprohibited solution that has not been visited by the current round of `SBTS` and is eligible for solution transition. Finally, for each visited solution during the search process, we fill up the corresponding positions in the three hash tables with 1. Figure 2 shows an illustrative example of a solution M being classified as tabu and hence being excluded for solution transition.

The main benefit of utilizing hash functions to maintain the tabu list is that we can quickly identify the tabu status of a neighboring solution in $\mathcal{O}(1)$ time. Furthermore, they eliminate the requirements to confine tabu status to a specific time period, relieving the algorithm designers of the time-consuming task of calibrating tabu tenure.

4. Experimental results

To evaluate the performance of the proposed `SBTS` algorithm, we show in this section the computational results in comparison with the best performing algorithms in the literature based on 100 well-known benchmark instances.

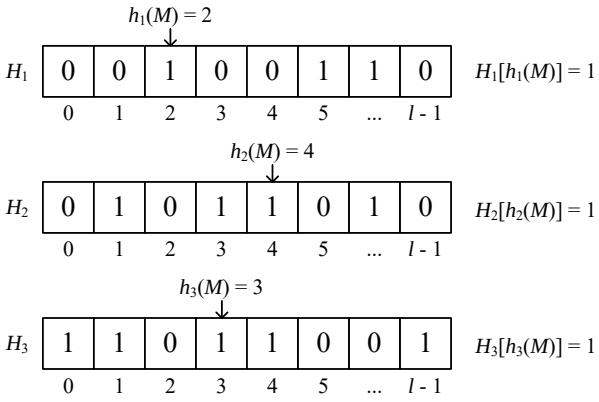


Figure 2: Illustrative example of the solution-based tabu strategy using three hash functions and the associated hash tables.

4.1. Benchmark sets

The computational assessments are based on four sets of 100 benchmark instances, which were previously used in various studies on dispersion (or diversity) problems (Lai et al., 2018; Martí et al., 2021a; Parreño et al., 2021; Wang et al., 2017; Zhou et al., 2017) and first adapted to the CDP in Martí et al. (2021b); Peiró et al. (2021). For each original instance, the capacity value of each node is generated at random in the range $[1, 1000]$. The total required capacity B is then calculated as the sum of all node's capacities multiplied by a factor φ of 0.2 and 0.3, obtaining two instances for each original one. The detailed information of these instances is described below.

- **GKD-b2 & GDK-b3** (40 instances): The data set GKD-b was initially generated by Martí *et al.* Martí et al. (2010) for the MDP, in which the distance d_{ij} between any two elements i and j is calculated as the Euclidean distances from randomly generated points with coordinates in $[0, 10]$. Then, 20 instances ($n = 50, 100$) are adapted by Peiró et al. (2021) for CDP then, as previously stated, two sets of instances with different node capacities and required capacity B is obtained for each original one.

- **GKD-c2 & GKD-c3** (20 instances): The data set GKD-c was first introduced by Duarte & Martí Duarte & Martí (2007) that contained large instances ($n = 500$) for MDP. In Peiró et al. (2021), the authors choose 10 instances and create two sets for each of them as previously mentioned.

- **SOM-a2 & SOM-a3** (20 instances): This data set was originally created by Martí *et al.* Martí et al. (2010), where the distances between any two elements were calculated using random values from an integer uniform distribution in $[0, 9]$. As in the previous sets, 10 suitable instances ($n = 50$) are selected and two sets of new instances are generated as before.

- **MDG-b2 & MDG-b3** (20 instances): This data set was first presented by Duarte & Martí Duarte & Martí (2007) and utilized in Gallego *et al.* Gallego et al. (2009), in which the distances between any two elements were real values in $[0, 1000]$. It is considered 10 large instances ($n = 500$) of the original MDG-b set obtaining two sets of new instances as before by Peiró et al. (2021).

$\varphi = 0.2$			$\varphi = 0.3$		
Benchmark set	Size	n	Benchmark set	Size	n
GKD-b2	20	50, 150	GKD-b3	20	50, 150
GKD-c2	10	500	GKD-c3	10	500
SOM-a2	10	50	SOM-a3	10	50
MDG-b2	10	500	MDG-b3	10	500

Table 1: List of benchmark sets.

Table 1 summarizes the main characteristics of the benchmarks listed above. Column 'Benchmark set' indicates the name of each set of instances, column 'Size' shows the number of instances of each benchmark set, and column 'n' refers to the number of nodes (elements). All of the benchmark instances are divided into two parts based on their capacity factors (φ), and are available for download[1].

4.2. Experimental settings

The proposed `SBTS` algorithm was programmed in C++[1] and complied using GNU g++ 8.3.0 with the '-O3' option. All experiments were carried out on a computer with an Intel Xeon E5-2695 v4 processor (2.10 GHz and 2GB RAM) running the Linux operating system.

For the comparative assessments, we adopt the best performing algorithms as our references: 1) the strategic oscillation method `SO` (Peiró et al., 2021), 2) the short-term scatter search `SS1` (Martí et al., 2021b), and 3) the long-term scatter search `SS2` (Martí et al., 2021b). These reference algorithms were run on an Intel Core i7 processor (2.80 GHz and 8GB RAM). Since the compared algorithms were tested on different computing platforms, we thus use CPU frequency (Chen et al., 2016; Zhou et al., 2022) to compare the speeds of the processors that were employed to test `SBTS` and the reference algorithms. Using our processor (Intel Xeon E5-2695 v4, 2.10 GHz) as a basis, the scaling factor of the processor (Intel Core i7, 2.80 GHz) used by the reference algorithms is 1.33, indicating that the processor used in our study is slightly slower. This time conversion is only made for indicative purposes, since the computing time of each algorithm is not only influenced by the processor, but also by some inaccessible factors such as the operating systems, compilers, and coding skills of the programmers. The authors of Peiró et al. (2021) and Martí et al. (2021b) generously shared the results of their algorithms, allowing us to conduct a direct and meaningful comparison. These algorithms also represent the state-of-the-art for tackling the CDP and together hold the best-known results for the available benchmark instances.

Considering the stochastic nature of our `SBTS` algorithm, each of the 100 benchmark instances was independently solved 40 times with different random seeds. The cutoff times of each run depend on two different stop criteria: 1) 10 seconds, and 2) 300 seconds. For convenience of presenting, we denote `SBTS` with 10 seconds as $\texttt{SBTS}_{\texttt{short}}$, and `SBTS` with 300 seconds as $\texttt{SBTS}_{\texttt{long}}$. Studying the outcomes of these two termination criteria reveal how `SBTS` performs when more computation time is available.

[1]The benchmark instances and the source code of the proposed `SBTS` algorithm will be made public at: https://github.com/hellozhilu/SBTS_CDP.

Parameter	Section	Description	Considered values	Final value
β_1	§3.5	In the first hash function $h_1(M)$	$[100, 200, 300, \ldots, 1000]$	300
β_2	§3.5	In the second hash function $h_2(M)$	$[100, 200, 300, \ldots, 1000]$	400
β_3	§3.5	In the third hash function $h_3(M)$	$[100, 200, 300, \ldots, 1000]$	500

Table 2: Setting of parameters.

4.3. Parameter tuning

The proposed SBTS algorithm requires three parameters β_1, β_2, and β_3 in each of the three hash functions (Section 3.5). To construct a suitable parameter calibration, we employ the IRACE package López-Ibáñez et al. (2016), which implements the Iterated F-race (IFR) method Birattari et al. (2010) and allows automatic parameter configuration. For the tuning experiment, we selected 40 hard instances from the benchmark sets GDK-c2, GDK-c3, MDG-b2, and MDG-b3. The tuning budget of IRACE was set to 1000 runs, and the cutoff times of each run was limited to 300 seconds. Table 2 shows the range of possible parameter values and final values provided by the IRACE package. The final values can be considered the default parameter setting of SBTS, which were consistently used in all of the experiments reported in this work. As shown in Section 4.4, SBTS achieves highly competitive results with this unique parameter setting. Generally, these parameters can be fine-tuned to obtain improved results, and such a practice is useful when one seeks the best possible solution for a given graph.

4.4. Computational results and comparisons with state-of-the-art algorithms

This section presents the computational results of the proposed SBTS algorithm ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$), in comparison with the three state-of-the-art algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) on the 100 benchmark instances.

Note that the results of the reference algorithms (SO, SS1, and SS2) were obtained from a single-run whereas our SBTS algorithms ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$) came from multiple-run (40 runs). Clearly, the best results favor multiple-run results. To make a fair comparison, we refer to average statistics when we compare single-run results with multiple-run results.

Table 3 summarizes the overall computational results while the detailed results are presented in Appendix A (Table A.1 and Table A.2). In Table 3, column 'Benchmark set' refers to the name of each set of instances while column 'n' indicates the number of nodes (elements). For each algorithm, the remaining columns show the average among all the results obtained in each set of the following statistics values:

- The best objective function value f_{best} found during 40 runs for $\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$. Then, column $\bar{f}_{best}$ contains the average of the best objective function values in each benchmark set.

- The average objective function value f_{avg} among the 40 runs for $\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$ (similarly as above, we denote as column $\bar{f}_{avg}$ the average of these values for each benchmark set).

- The worst objective function value f_{wst} found during 40 runs for $\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$ (column $\bar{f}_{wst}$).

  In the case of a single-run, i.e., SO, SS1, and SS2 algorithms, these measures are identical, then for the sake of simplicity, we denote as $\bar{f}_{avg}$ the average of the best objective function values obtained in each benchmark set.

Benchmark set	n	SO Peiró et al. (2021) (1 run) $\bar{f}_{avg}$	$DevB$	$DevO$	SS1 Martí et al. (2021b) (1 run) $\bar{f}_{avg}$	$DevB$	$DevO$	SS2 Martí et al. (2021b) (1 run) $\bar{f}_{avg}$	$DevB$	$DevO$
GKD-b2	50	**112.33**	0.0%	0.0%	111.72	0.5%	0.5%	111.72	0.5%	0.5%
	150	115.15	2.9%	3.0%	117.46	1.0%	1.1%	117.62	0.8%	0.9%
GKD-c2	500	7.46	20.6%	20.6%	9.17	2.3%	2.4%	9.17	2.3%	2.4%
SOM-a2	50	**4.10**	0.0%	0.0%	**4.10**	0.0%	0.0%	**4.10**	0.0%	0.0%
MDG-b2	500	41.15	31.9%	-	43.51	28.0%	-	48.10	20.4%	-
GKD-b3	50	96.95	0.9%	0.9%	97.58	0.2%	0.2%	97.62	0.2%	0.2%
	150	103.42	4.2%	4.3%	107.03	0.9%	1.0%	106.87	1.1%	1.1%
GKD-c3	500	6.49	22.4%	22.5%	8.20	1.9%	2.0%	8.20	1.9%	2.0%
SOM-a3	50	1.50	28.6%	28.6%	1.90	9.5%	9.5%	2.00	4.8%	4.8%
MDG-b3	500	11.09	61.4%	-	21.80	24.1%	-	22.21	22.7%	-
	Average	49.96	17.3%	10.0%	52.25	6.9%	2.1%	52.76	5.5%	1.5%

Benchmark set	n	$\text{SBTS}_{\text{short}}$ (40 runs) $\bar{f}_{best}$	$\bar{f}_{avg}$	$\bar{f}_{wst}$	$DevB$	$DevO$	$\text{SBTS}_{\text{long}}$ (40 runs) $\bar{f}_{best}$	$\bar{f}_{avg}$	$\bar{f}_{wst}$	$DevB$	$DevO$
GKD-b2	50	**112.33**	**112.33**	**112.33**	0.0%	0.0%	**112.33**	**112.33**	**112.33**	0.0%	0.0%
	150	118.56	118.39	117.94	0.2%	0.3%	**118.60**	118.46	118.39	0.1%	0.2%
GKD-c2	500	9.28	9.19	8.99	2.1%	2.2%	**9.39**	9.34	9.30	0.5%	0.6%
SOM-a2	50	**4.10**	**4.10**	**4.10**	0.0%	0.0%	**4.10**	**4.10**	**4.10**	0.0%	0.0%
MDG-b2	500	55.41	47.04	38.34	22.2%	-	**60.46**	57.07	50.81	5.6%	-
GKD-b3	50	**97.79**	**97.79**	**97.79**	0.0%	0.0%	**97.79**	**97.79**	**97.79**	0.0%	0.0%
	150	**108.01**	107.59	107.37	0.4%	0.5%	**108.01**	107.76	107.57	0.2%	0.3%
GKD-c3	500	8.32	8.23	8.15	1.6%	1.7%	**8.36**	8.34	8.29	0.3%	0.4%
SOM-a3	50	**2.10**	2.08	1.80	0.8%	0.8%	**2.10**	**2.10**	**2.10**	0.0%	0.0%
MDG-b3	500	26.06	22.66	18.89	21.1%	-	**28.72**	26.12	22.20	9.1%	-
	Average	54.20	52.94	51.57	4.8%	0.7%	**54.99**	54.34	53.29	**1.6%**	**0.2%**

*0.0 means less than 0.0001

Table 3: Summary results reported by the proposed SBTS algorithm ($\text{SBTS}_{\text{short}}$ and $\text{SBTS}_{\text{long}}$) and the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b), over each benchmark set.

– The percent deviation of the average objective function value with respect the best solution found in the experiment (column $DevB$).

– The average deviation of the average solution from the optimal solution (when available) (column $DevO$).

The average results of each indicator provided by each compared algorithm are reported in the last row ('Average'). For each benchmark set, the best of each indicator value among all compared algorithms is highlighted in boldface.

From Table 3, we observe that our $\text{SBTS}_{\text{short}}$ algorithm performs very well on all 100 benchmark instances and shows its superiority over the reference algorithms. In terms of the average solution quality ($\bar{f}_{avg}$), $\text{SBTS}_{\text{short}}$ achieves a lower average deviation of 0.7% from the optimal solution, which is 2.14 times, 3.00 times, and 14.29 times less than the 1.5% of SS2, the 2.1% of SS1, and the 10.0% of SO, respectively. The average result of $\text{SBTS}_{\text{short}}$ is 52.94, which is marginally (1.00, 1.01, and 1.06 times) higher than the 52.76 of SS2, the 52.25 of SS1, and the 49.96 of SO. By increasing the $\text{SBTS}_{\text{short}}$ termination criterion of 10 seconds to 300 seconds, $\text{SBTS}_{\text{long}}$ achieves a still better performance, always obtaining equal or better results

Benchmark set	n	SO (1 run) #Best	SS1 (1 run) #Best	SS2 (1 run) #Best	$\texttt{SBTS}_{\texttt{short}}$ (40 runs) #Best	$\texttt{SBTS}_{\texttt{short}}$ (40 runs) #Avg.	$\texttt{SBTS}_{\texttt{long}}$ (40 runs) #Best	$\texttt{SBTS}_{\texttt{long}}$ (40 runs) #Avg.
GKD-b2	50	10	8	8	10	10	10	10
	150	0	0	0	7	2	7	5
GKD-c2	500	0	0	0	0	0	9	2
SOM-a2	50	10	10	10	10	10	10	10
MDG-b2	500	0	0	0	0	0	10	0
GKD-b3	50	6	7	8	10	10	10	10
	150	0	0	0	8	2	8	4
GKD-c3	500	0	0	0	5	0	9	2
SOM-a3	50	5	8	9	10	7	10	10
MDG-b3	500	0	0	0	0	0	10	0
	Total	31	33	35	60	41	93	53

Table 4: Statistical results for each compared algorithms (SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), SS2 (Martí et al., 2021b), $\texttt{SBTS}_{\texttt{short}}$, and $\texttt{SBTS}_{\texttt{long}}$) over each benchmark set, the number of best and average solutions that match or improve on the best-known solutions (#Best and #Avg.).

in terms of the average solution quality ($\bar{f}_{avg}$). Particularly, $\texttt{SBTS}_{\texttt{long}}$ achieves the smallest deviation of 0.2% from the optimal solution, 7.50 times, 10.50 times, and 50.00 times less than SS2, SS1, and SO, respectively. The average results of $\texttt{SBTS}_{\texttt{long}}$ is 54.34, which is 1.03, 1.04, and 1.09 times higher than SS2, SS1, and SO, respectively.

The computation time of our SBTS algorithm is fairly competitive. Recall that the indicated time for the reference algorithms are scaled for our computer, and that the average time of a multiple-run algorithm can be compared to the time of a single-run algorithm. On average the CPU time of SO, SS1, and SS2 is 17.07, 1.55, and 2.30 seconds, respectively. The average running time per run in seconds to reach the final solution is 3.28 and 50.59 for $\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$, respectively. These results shows that $\texttt{SBTS}_{\texttt{short}}$ is not significantly slower than any of the reference algorithms, and generally requires comparable computing time. By extending the stop condition to 300 seconds, $\texttt{SBTS}_{\texttt{long}}$, which finds better solutions, consumes more computing time than $\texttt{SBTS}_{\texttt{short}}$ as expected.

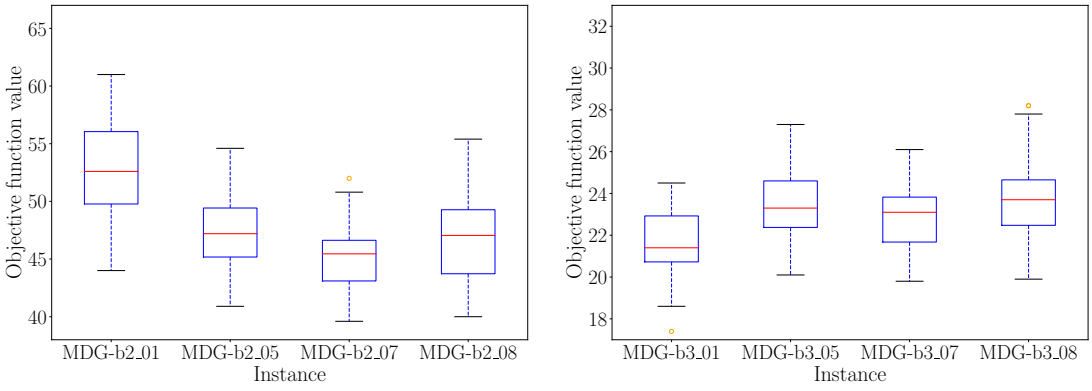
Table 4 counts for each compared algorithm over each benchmark set, the number of best solutions f_{best} or average values f_{avg} that match or improve the best-known solutions (#Best and #Avg., respectively). Both values match for the single-run algorithms. As shown in Table 4, our $\texttt{SBTS}_{\texttt{short}}$ algorithm matches or improves the best-known solutions for 60 out of 100 instances, while $\texttt{SBTS}_{\texttt{long}}$ performs better in 93 out of 100 instances. As previously stated, we should look at average statistics when comparing multiple-run algorithms to single-run algorithms. According to the average indicator #Avg., our SBTS algorithms ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$) are still the best-performing algorithms among all three reference algorithms, matching or improving 41 and 53 out of 100 instances.

To validate the above findings, Table 5 shows the p-values from the Wilcoxon signed-rank test with a confidence level of 99% of SBTS versus all reference algorithms (SO, SS1, and SS2) in terms of the average solution quality. A p-value smaller than 0.01 indicates that the two compared algorithms are significantly different. As indicated in Table 5, $\texttt{SBTS}_{\texttt{short}}$ performs significantly better than the reference algorithms on all 100 benchmark instances. This conclusion remains valid for $\texttt{SBTS}_{\texttt{long}}$ since it always performs better than $\texttt{SBTS}_{\texttt{short}}$.

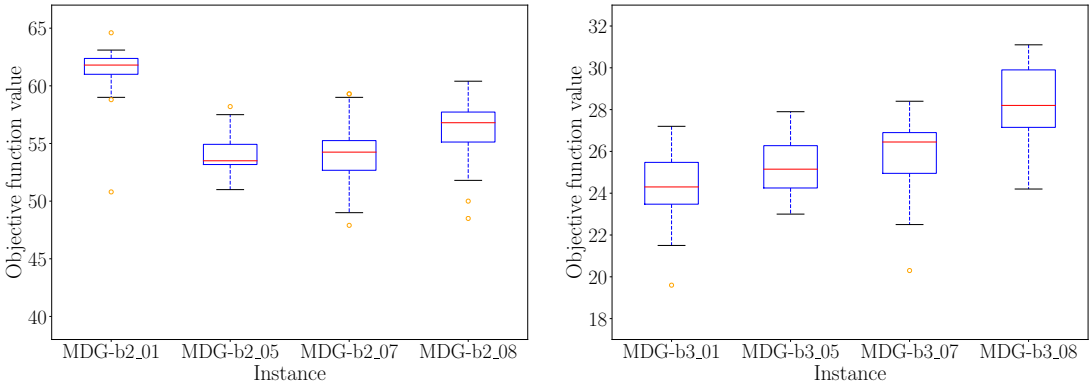
To test the robustness of SBTS, Figure 3 depicts the box plots of $\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$ on 8 representative hard instances randomly selected from the large benchmark sets MDG-b2

Benchmark set	SBTS *vs.* SO		SBTS *vs.* SS1		SBTS *vs.* SS2	
	$\text{SBTS}_{\text{short}}$	$\text{SBTS}_{\text{long}}$	$\text{SBTS}_{\text{short}}$	$\text{SBTS}_{\text{long}}$	$\text{SBTS}_{\text{short}}$	$\text{SBTS}_{\text{long}}$
50 instances with $\varphi = 0.2$	5.96e-06	1.73e-06	2.69e-05	7.71e-07	4.65e-01	7.71e-07
50 instances with $\varphi = 0.3$	5.21e-08	5.20e-08	3.15e-05	5.85e-07	1.70e-03	1.24e-06

Table 5: Statistical results (p-values) from the Wilcoxon signed-rank test with a confidence level of 99% of the proposed SBTS algorithms ($\text{SBTS}_{\text{short}}$ and $\text{SBTS}_{\text{long}}$) versus the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) over total benchmark sets.



(a) The box-plots of $\text{SBTS}_{\text{short}}$ on 8 instances from sets MDG-b2 and MDG-b3.



(b) The box-plots of $\text{SBTS}_{\text{long}}$ on 8 instances from sets MDG-b2 and MDG-b3.

Figure 3: The box-plots of the proposed SBTS algorithms ($\text{SBTS}_{\text{short}}$ and $\text{SBTS}_{\text{long}}$) on 8 representative hard instances from benchmark sets MDG-b2 and MDG-b3.

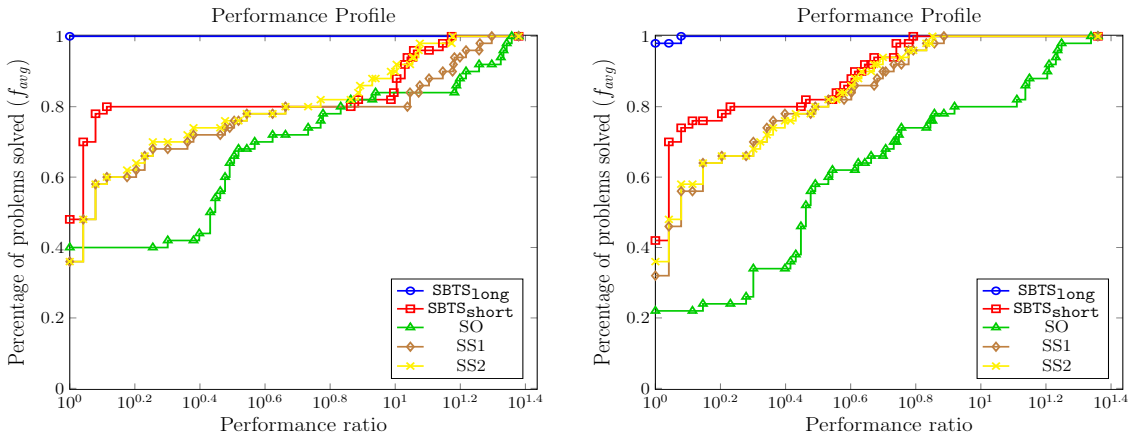


Figure 4: Performance profiles (f_{avg}) of the proposed SBTS algorithms (SBTS$_{short}$ and SBTS$_{long}$) and the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) on the 100 benchmark instances (left: 50 instances with $\varphi = 0.2$, right: 50 instances with $\varphi = 0.3$).

and MDG-b3. From Figure 3, we observe that even in short termination criterion (10 seconds), SBTS$_{short}$ obtains similar solutions when replicated. As expected, when given more computing time (300 seconds), SBTS$_{long}$ becomes more stable, and is able to further improve its results.

To complement the above findings, we provide a global performance evaluation of all compared algorithms in a visual way with performance profiles (Dolan & Moré, 2002). Since the performance profile is defined for minimization problems, when dealing with a maximization problem, the normalization to the minimization problem was done by $f'_{a,i} = \max\{f_{a,i} : a \in \mathcal{A}, i \in \mathcal{I}\} - f_{a,i} + 1$, where $f_{a,i}$ is the (average) objective function value obtained after solving problem i by algorithm a. In particular, we define the performance ratio by $r_{a,i} = f'_{a,i} / \min\{f'_{a,i} : a \in \mathcal{A},\ i \in \mathcal{I}\}$, to compare a set of algorithms $\mathcal{A}$ over a set of instances $\mathcal{I}$. We just set $r_{a,i} = +\infty$ if an algorithm a fails to solve an instance i. The performance function of an algorithm a is thus given by $\rho_a(\tau) = |\{i \in \mathcal{I} : r_{a,i} \leq \tau\}|/|\mathcal{I}|$, which computes the fraction of instances that an algorithm a can solve with at most τ many times the objective function value of the best algorithm. $\rho_a(1)$ denotes the percentage of instances that an algorithm a solved better than, or as better as the other algorithms in $\mathcal{A}$ do. For a large enough τ value ($\tau = r_f$), $\rho_a(r_f)$ corresponds to the maximum percentage of instances that an algorithm a can solve. The efficiency and robustness of a are measured by the quantities $\rho_a(1)$ and $\rho_a(r_f)$, respectively. Figure 4 depicts the performance profiles in terms of the average solution (by using the software *perprof-py* by Siqueira et al. (2016)) of SBTS (SBTS$_{short}$ and SBTS$_{long}$) and the three reference algorithms (SO, SS1, and SS2) on the 100 benchmark instances.

From Figure 4, we find that SBTS has an excellent performance, surpassing all reference algorithms in terms of the average solution for all 100 benchmark instances. Specifically, SBTS$_{long}$ reports the highest $\rho_a(1)$ value, indicating that it can quickly find the highest objective function values for the tested instances. SBTS$_{long}$ has strong robustness since it is the first to reach $\rho_a(r_f)$, which means it can consistently solve all the instances.

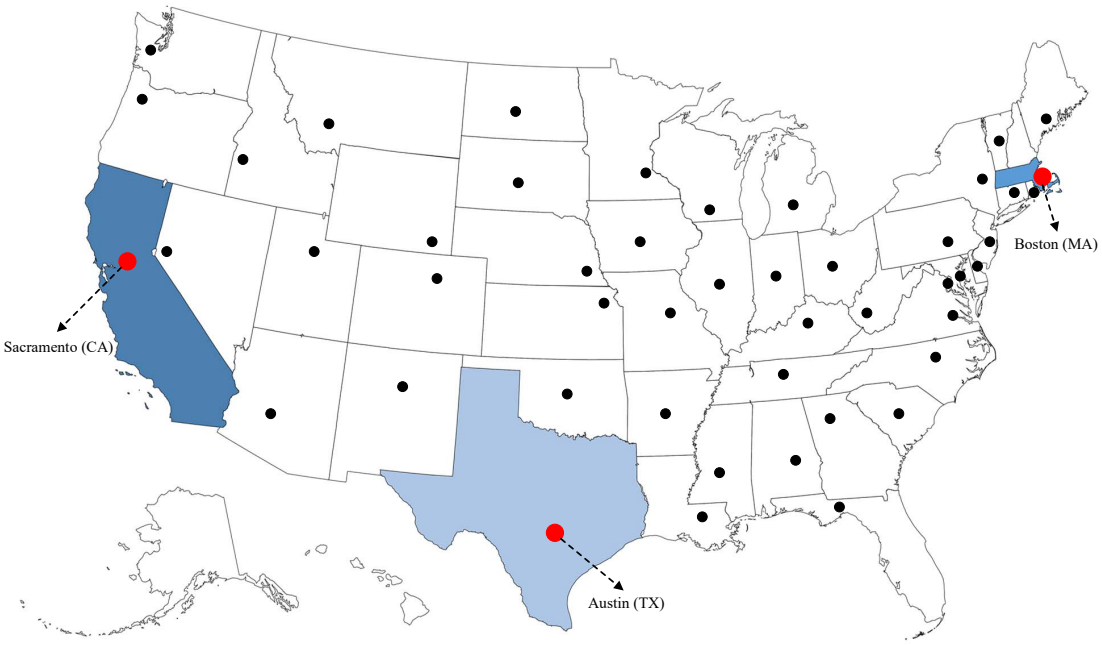


Figure 5: A map of the U.S. with 49 potential sites (black points) considered by the company to establish new branch office. Our `SBTS` algorithm provides the company with the decision of opening new offices (red points) in Sacramento (CA), Austin (TX), and Boston (MA) .

5. A real-world application

This section considers a real case on the 49-node data set first described in Daskin (2011), and later studied in Lozano-Osorio et al. (2022), consisting of the capitals of the continental United States plus Washington, DC, where an international branch company wants to expand its operations by opening new offices in the continental U.S.. Here, we adapt it to obtain a real-world application for the capacitated dispersion problem.

In this application, a company that provides a specific service through a network of sites aims to expand internationally by opening new branch offices in some of the 49 U.S. cities to launch novel product lines in this competitive market. Specifically, each city i has a capacity of service c_i that is determined by the customer demands in each city, i.e., the number of customers that can be attended in the potential site at city i, which is proportional to their population. The distance between sites i and j is known as d_{ij}. The company wants to provide service for at least 5000 customers in all their new offices. Thus, our goal is to select the sites as disperse as possible in the U.S. while meets the constraint of minimum customer demands to increase visibility for the new affiliates and create new market opportunities in the U.S..

Figure 5 shows a map of the U.S. with 49 potential sites (black points) considered by the company to establish new branch office. Our `SBTS` algorithm helps the company make the best strategical decision to increase their global influence by opening new offices in Sacramento (CA), Austin (TX), and Boston (MA) (the corresponding states are depicted with blue colors while the cities are with red points in the figure). From Figure 5, we observe that opening branch offices in Sacramento (CA), Austin (TX), and Boston (MA) can meet the needs of customers in the western, central, and eastern regions of the U.S., respectively. In a word,

		Random construction procedure				**Greedy construction heuristic**			
Benchmark set	n	$\bar{f}_{best}$	$DevB$	$DevO$	$t(s)$	$\bar{f}_{best}$	$DevB$	$DevO$	$t(s)$
GKD-b2	50	81.37	22.6%	27.6%	0.00	**105.09**	0.0%	6.4%	0.00
	150	80.02	29.5%	32.6%	0.00	**113.53**	0.0%	4.4%	0.00
GKD-c2	500	5.06	41.8%	46.2%	0.00	**8.69**	0.0%	7.6%	0.14
SOM-a2	50	0.40	88.6%	90.2%	0.00	**3.50**	0.0%	14.6%	0.00
MDG-b2	500	0.92	97.1%	-	0.00	**32.24**	0.0%	-	0.09
GKD-b3	50	70.50	23.8%	27.9%	0.00	**92.47**	0.0%	5.4%	0.00
	150	72.42	29.7%	33.0%	0.00	**103.06**	0.0%	4.7%	0.00
GKD-c3	500	4.69	39.8%	44.0%	0.00	**7.79**	0.0%	6.9%	0.24
SOM-a3	50	0.00	100.0%	100.0%	0.00	**1.70**	0.0%	19.0%	0.00
MDG-b3	500	0.31	98.1%	-	0.00	**16.33**	0.0%	-	0.21
	Average	31.57	57.1%	50.2%	0.00	**48.44**	**0.00%**	**8.6%**	0.07

*0.00 means less than 0.001

Table 6: Summary results of our proposed greedy construction heuristic against the random construction procedure over each benchmark set.

these results perfectly match the company interests in expanding their global market and gaining a large number of customers.

6. Analysis

In this section, we focus on two key components of the proposed `SBTS` algorithm: 1) the initial solution procedure utilizing a greedy construction heuristic, and 2) the solution-based tabu search strategy that uses three hash functions to determine the tabu status of neighboring solutions. We will analyze and discuss the effect of these two important components to the performance of `SBTS`.

6.1. Computational results of the greedy construction heuristic

To evaluate the performance of the initial solution procedure of `SBTS`, this section compares our greedy construction heuristic against a random construction procedure based on the 100 benchmark instances. Each construction procedure was independently executed 40 times per instance, with a cutoff time as indicated in Section 4.2 per run. It is worth mentioning that these construction procedures stop once the total required capacity B is met, and reports the same information as Section 4.4. Table 6 summarizes the computational results of our proposed greedy construction heuristic versus the random construction procedure over each benchmark set.

From Table 6, we observe that our greedy construction heuristic outperforms the random construction procedure for all 100 benchmark instances. In terms of the best solution quality, the greedy construction heuristic obtains a lower average deviation of 8.6% from the optimal solution, which is 5.84 times less than the 50.2% of the random construction procedure. The average results of the greedy construction heuristic is 48.44, which is 1.53 times higher than the 31.57 of the random construction procedure. This experiment shows that our greedy construction heuristic provide an initial solution of good quality for `SBTS`.

6.2. Computational results of the solution-based tabu search strategy

The solution-based tabu search strategy uses hash tables and hash functions to implement the tabu list. To assess the importance of this strategy, we create an attribute-based tabu

Benchmark set	n	Attribute-based tabu search (ABTS) $\bar{f}_{best}$	$DevB$	$DevO$	$t(s)$	**Solution-based tabu search (SBTS)** $\bar{f}_{best}$	$DevB$	$DevO$	$t(s)$
GKD-b2	50	112.33	0.0%	0.0%	0.41	**112.33**	0.0%	0.0%	0.54
	150	115.73	2.4%	2.5%	0.01	**118.61**	0.0%	0.1%	3.34
GKD-c2	500	9.07	2.8%	3.5%	11.11	**9.33**	0.0%	0.7%	13.38
SOM-a2	50	4.00	2.4%	2.4%	8.77	**4.10**	0.0%	0.0%	1.21
MDG-b2	500	48.07	19.9%	-	128.23	**60.04**	0.0%	-	183.77
GKD-b3	50	97.79	0.0%	0.0%	0.68	**97.79**	0.0%	0.0%	1.04
	150	105.17	2.6%	2.7%	0.96	**107.98**	0.0%	0.1%	7.31
GKD-c3	500	8.15	1.8%	2.6%	13.86	**8.30**	0.0%	0.8%	17.52
SOM-a3	50	1.90	9.5%	9.5%	6.75	**2.10**	0.0%	0.0%	3.21
MDG-b3	500	25.60	10.5%	-	126.44	**28.61**	0.0%	-	204.17
	Average	52.78	5.2%	2.9%	29.72	**54.92**	**0.0%**	**0.2%**	43.55

*0.0 means less than 0.01

Table 7: Summary results of our proposed solution-based tabu search procedure against the attribute-based tabu search procedure over each benchmark set.

search procedure where the tabu list only records the performed moves. This experiment follows the same experimental protocol and reports the same information as Section 6.1. Table 7 summarizes the computational results of our proposed solution-based tabu search (SBTS) procedure versus the attribute-based tabu search (ABTS) procedure over each benchmark set.

As observed in Table 7, SBTS achieves a lower average deviation of 0.2% from the optimal solution, which is 14.50 times less than the 2.9% of ABTS in terms of the best solution. The average result of SBTS is 54.92, which is slightly (1.04 times) higher than the 52.78 of ABTS. This experiment confirms the usefulness of the solution-based tabu search strategy, which positively contributes to the high performance of our `SBTS` algorithm.

7. Conclusions and future work

We had two main objectives for this work, a methodological and a practical objective. In the former one, we experiment in the implementation of the less studied solution-based tabu search (`SBTS`) methodology on a max-min problem, the capacitated dispersion problem (CDP). To the latter objective, we tested our procedure to a realistic case, and we developed a state-of-the-art procedure for the CDP.

In the methodological framework, our `SBTS` method features a greedy construction heuristic to generate an initial solution of good quality. Moreover, the joint use of three hash functions speeds up the process of identifying the tabu status of candidate solutions. Finally, the definition of a large neighborhood by the combination of three neighborhoods enhances the solution quality and the computational efficiency. Furthermore, to shed light on the impacts of the initial greedy construction heuristic and the solution-based tabu strategy on the performance of our `SBTS` proposal, we presented additional experiments to investigate these two key components. In particular, the comparison between the ABTS and SBTS procedures confirms that the solution-based strategy is a key feature in our heuristic.

Finally, the computational results on four sets of 100 well-known benchmark instances indicate that `SBTS` competes very favorably with the state-of-the-art algorithms in the literature. Specifically, `SBTS` finds the optimal results for 73 out of the 80 instances for which it is known,

and it improves the best-known results for the remaining 20 instances with unknown optimal solutions.

Regarding the applicability, we additionally presented an application of our approach to deal with the realistic location problem, where an international branch company wants to open new offices in the U.S. to expand its business. We also make the source code of our `SBTS` algorithm publicly available, which can be used by researchers and practitioners to solve a variety of practical applications that can be formulated as the CDP and the related dispersion (or diversity) problems.

For future research, several promising areas deserve attention of researchers. First, as inspired by a recent review of Martí et al. (2022), many heuristics have been proposed for dispersion (or diversity) problems, but only a few exact methods have been investigated. We may focus on these exact methods in the future to solve the CDP to optimality on more hard instances. As a complement to exact methods, another idea is to design more powerful metaheuristics that can find already known optimal solutions in a short running time, for example, trajectory-based metaheuristics like iterated greedy or variable neighborhood search (Ruiz & Stützle, 2007; Mladenović et al., 2016), or population-based algorithms like memetic or genetic algorithms (Neri & Cotta, 2012; Tan et al., 2014). Second, the ideas of solution-based tabu search are rather general, and they may be usefully applied to solve other related dispersion (or diversity) problems as well as similar combinatorial optimization problems.

Acknowledgments. We are grateful to the reviewers for their helpful comments and suggestions. This research was partially supported by the National Natural Science Foundation of China (Grant no. 72101149, Grant no. 71871144, and Grant no. 61703213), the Shanghai Pujiang Program (Grant no. 22PJC080), the Spanish Ministerio de Ciencia e Innovación with grant ref. PID2021-125709OB-C21 funded by MCIN /AEI /10.13039/501100011033 / FEDER, UE, the Generalitat Valenciana (CIAICO/2021/224), and the six talent peaks project in Jiangsu Province of China (Grant no. RJFW-011).

References

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. *Experimental methods for the analysis of optimization algorithms*, (pp. 311–336).

Chen, Y., Hao, J.-K., & Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, *253*, 25–39.

Daskin, M. S. (2011). *Network and discrete location: models, algorithms, and applications*. John Wiley & Sons.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*, 201–213.

Duarte, A., & Martí, R. (2007). Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, *178*, 71–84.

Galinier, P., & Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, *3*, 379–397.

Gallego, M., Duarte, A., Laguna, M., & Martí, R. (2009). Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, *44*, 411.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

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

Laguna, M., Marti, R., & Campos, V. (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research*, *26*, 1217–1230.

Lai, X., & Fu, Z.-H. (2019). A tabu search approach with dynamical neighborhood size for solving the maximum min-sum dispersion problem. *IEEE Access*, *7*, 181357–181368.

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

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

Lai, X., Yue, D., Hao, J.-K., & Glover, F. (2018). Solution-based tabu search for the maximum min-sum dispersion problem. *Information Sciences*, *441*, 79–94.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Lozano-Osorio, I., Martínez-Gavara, A., Martí, R., & Duarte, A. (2022). Max–min dispersion with capacity and cost for a practical location problem. *Expert Systems with Applications*, *200*, 116899.

Lu, Z., Hao, J.-K., & Wu, Q. (2020). A hybrid evolutionary algorithm for finding low conductance of large graphs. *Future Generation Computer Systems*, *106*, 105–120.

Lu, Z., Hao, J.-K., & Zhou, Y. (2019). Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem. *Computers & Operations Research*, *111*, 43–57.

Martí, R., Martínez-Gavara, A., Pérez-Peló, S., & Sánchez-Oro, J. (2022). A review on discrete diversity and dispersion maximization from an OR perspective. *European Journal of Operational Research*, *299*, 795–813.

Martí, R., Duarte, A., Martínez-Gavara, A., & Sánchez-Oro, J. (2021a). The MDPLIB 2.0 Library of benchmark instances for diversity problems. URL: https://www.uv.es/rmarti/paper/mdp.html.

Martí, R., Gallego, M., & Duarte, A. (2010). A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, *200*, 36–44.

Martí, R., Gallego, M., Duarte, A., & Pardo, E. G. (2013). Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, *19*, 591–615.

Martí, R., Laguna, M., & Glover, F. (2006). Principles of scatter search. *european Journal of operational Research*, *169*, 359–372.

Martí, R., Martínez-Gavara, A., & Sánchez-Oro, J. (2021b). The capacitated dispersion problem: an optimization model and a memetic algorithm. *Memetic Computing*, *13*, 131–146.

Martínez-Gavara, A., Campos, V., Laguna, M., & Martí, R. (2017). Heuristic solution approaches for the maximum minsum dispersion problem. *Journal of Global Optimization*, *67*, 671–686.

Martínez-Gavara, A., Corberán, T., & Martí, R. (2021). Grasp and tabu search for the generalized dispersion problem. *Expert Systems with Applications*, *173*, 114703.

Mladenović, N., Todosijević, R., & Urošević, D. (2016). Less is more: basic variable neighborhood search for minimum differential dispersion problem. *Information Sciences*, *326*, 160–171.

Neri, F., & Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, *2*, 1–14.

Parreño, F., Álvarez-Valdés, R., & Martí, R. (2021). Measuring diversity. A review and an empirical analysis. *European Journal of Operational Research*, *289*, 515–532.

Peiró, J., Jiménez, I., Laguardia, J., & Martí, R. (2021). Heuristics for the capacitated dispersion problem. *International Transactions in Operational Research*, *28*, 119–141.

Porumbel, D. C., Hao, J. K., & Glover, F. (2011). A simple and effective algorithm for the MaxMin diversity problem. *Annals of Operations Research*, *186*, 275–293.

Prokopyev, O. A., Kong, N., & Martinez-Torres, D. L. (2009). The equitable dispersion problem. *European Journal of Operational Research*, *197*, 59–67.

Rosenkrantz, D. J., Tayi, G. K., & Ravi, S. (2000). Facility dispersion problems under capacity and cost constraints. *Journal of Combinatorial Optimization*, *4*, 7–33.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*, 2033–2049.

Siqueira, A. S., da Silva, R. C., & Santos, L.-R. (2016). Perprof-py: A python package for performance profile of mathematical optimization software. *Journal of Open Research Software*, *4*.

Tan, G. W.-H., Ooi, K.-B., Leong, L.-Y., & Lin, B. (2014). Predicting the drivers of behavioral intention to use mobile learning: A hybrid sem-neural networks approach. *Computers in Human Behavior*, *36*, 198–213.

Wang, Y., Hao, J.-K., Glover, F., & Lü, Z. (2014). A tabu search based memetic algorithm for the maximum diversity problem. *Engineering Applications of Artificial Intelligence*, *27*, 103–114.

Wang, Y., Lü, Z., & Su, Z. (2021). A two-phase intensification tabu search algorithm for the maximum min-sum dispersion problem. *Computers & Operations Research*, *135*, 105427.

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

Wei, Z., & Hao, J.-K. (2021). Multistart solution-based tabu search for the set-union knapsack problem. *Applied Soft Computing*, *105*, 107260.

Zhou, Q., Hao, J.-K., & Wu, Q. (2022). A hybrid evolutionary search for the generalized quadratic multiple knapsack problem. *European Journal of Operational Research*, *296*, 788–803.

Zhou, Y., Hao, J.-K., & Duval, B. (2017). Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation*, *21*, 731–745.

A. Appendix

The appendix presents instance-by-instance results of the proposed SBTS algorithms ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$) and the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) on the 100 benchmark instances. The detailed results are summarized in Table A.1 (50 instances with $\varphi = 0.2$) and Table A.2 (50 instances with $\varphi = 0.3$).

In Table A.1 and Table A.2, column 'Instance ($\varphi = 0.2$)' and 'n' indicate the name (capacity factor) and the number of nodes (elements) for each instance. Column 'f_{opt}' gives the optimal solution obtained by Gurobi from the literature Martí et al. (2021b). The remaining columns show the results reached by each compared algorithm (SO, SS1, SS2, $\texttt{SBTS}_{\texttt{short}}$, and $\texttt{SBTS}_{\texttt{long}}$): the best objective function value (column 'f_{best}'), the average objective function value (column 'f_{avg}'), the worst objective function value (column 'f_{wst}'), and the average computation time per run in seconds to reach the final solution, and a time less than 0.01 seconds is indicated as 0 (column '$t(s)$'). Note that SBTS ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$) conducts multiple-run (40 runs) per instance and reports all of the above indicators while the three reference algorithms perform single-run and report f_{best} and $t(s)$. A boldface denotes an optimal solution reached by the algorithm.

Instance ($\varphi = 0.2$)	n	f_{opt}	SO (1 run)		SS1 (1 run)		SS2 (1 run)		$\texttt{SBTS}_{\texttt{short}}$ (40 runs)				$\texttt{SBTS}_{\texttt{long}}$ (40 runs)			
			f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	f_{wst}	$t(s)$	f_{best}	f_{avg}	f_{wst}	$t(s)$
GKD-b_11_n50_m5	50	147.2	**147.2**	0	**147.2**	0	**147.2**	0	**147.2**	147.2	147.2	0.7	**147.2**	147.2	147.2	0.4
GKD-b_12_n50_m5	50	178.1	**178.1**	0	**178.1**	0	**178.1**	0	**178.1**	178.1	178.1	0.7	**178.1**	178.1	178.1	0.6
GKD-b_13_n50_m5	50	96.1	**96.1**	0	93.6	0	93.6	0	**96.1**	96.1	96.1	0.8	**96.1**	96.1	96.1	0.6
GKD-b_14_n50_m5	50	84.6	**84.6**	0	**84.6**	0	**84.6**	0	**84.6**	84.6	84.6	0.7	**84.6**	84.6	84.6	0.6
GKD-b_15_n50_m5	50	154.9	**154.9**	0	**154.9**	0	**154.9**	0	**154.9**	154.9	154.9	0.7	**154.9**	154.9	154.9	0.6
GKD-b_16_n50_m15	50	77.7	**77.7**	0	**77.7**	0	**77.7**	0	**77.7**	77.7	77.7	0.6	**77.7**	77.7	77.7	0.6
GKD-b_17_n50_m15	50	41.8	**41.8**	0	38.2	0	38.2	0	**41.8**	41.8	41.8	0.8	**41.8**	41.8	41.8	0.6
GKD-b_18_n50_m15	50	108.5	**108.5**	0	**108.5**	0	**108.5**	0	**108.5**	108.5	108.5	0.8	**108.5**	108.5	108.5	0.5
GKD-b_19_n50_m15	50	119.1	**119.1**	0	**119.1**	0	**119.1**	0	**119.1**	119.1	119.1	0.8	**119.1**	119.1	119.1	0.2
GKD-b_20_n50_m15	50	115.3	**115.3**	0	**115.3**	0	**115.3**	0	**115.3**	115.3	115.3	0.5	**115.3**	115.3	115.3	0.3
GKD-b_41_n150_m15	150	164.2	158.8	0	161.8	0.2	161.8	0.1	163.9	163.7	162.6	3.1	164.1	163.8	163.8	2.9
GKD-b_42_n150_m15	150	84.3	83.3	1.0	84.1	0.1	84.1	0.1	**84.3**	84.3	84.3	1.6	**84.3**	84.3	84.3	1.0
GKD-b_43_n150_m15	150	63.3	58.9	0	62.0	0.1	62.0	0.1	**63.3**	63.3	63.1	2.5	**63.3**	63.3	63.3	2.0
GKD-b_44_n150_m15	150	103.3	99.0	1.0	101.6	0.1	101.6	0.1	102.2	102.2	102.2	1.3	102.2	102.2	102.2	0.5
GKD-b_45_n150_m15	150	106.6	104.3	0	104.4	0.1	105.8	0.2	**106.6**	106.5	105.2	2.4	**106.6**	106.6	106.6	2.0
GKD-b_46_n150_m45	150	124.5	118.7	0	123.8	0.1	123.8	0.1	**124.5**	124.4	123.2	1.4	**124.5**	124.5	124.5	1.2
GKD-b_47_n150_m45	150	163.4	158.2	0	162.4	0.1	162.6	0.2	163.1	162.9	162.7	3.3	163.3	163.1	162.8	13.2
GKD-b_48_n150_m45	150	100.2	98.1	1.0	98.7	0.1	98.7	0.1	**100.2**	100.0	99.7	4.5	**100.2**	100.1	99.8	8.0
GKD-b_49_n150_m45	150	166.3	164.5	0	165.5	0.1	165.5	0.1	**166.3**	166.3	166.3	0.6	**166.3**	166.3	166.3	0.3
GKD-b_50_n150_m45	150	111.2	107.7	1.0	110.3	0.1	110.3	0.1	**111.2**	110.3	110.1	2.8	**111.2**	110.4	100.3	1.2
GKD-c_01_n500_m50	500	9.4	7.2	18.0	9.2	3.0	9.2	3.4	9.3	9.2	9.0	4.3	9.3	9.3	9.3	14.2
GKD-c_02_n500_m50	500	9.5	7.6	15.0	9.3	3.2	9.3	3.3	9.4	9.3	9.1	5.6	**9.5**	9.5	9.5	51.4
GKD-c_03_n500_m50	500	9.4	7.2	22.0	9.2	2.9	9.2	4.2	9.3	9.2	9.1	4.7	**9.4**	9.4	9.4	43.5
GKD-c_04_n500_m50	500	9.3	7.5	15.0	9.1	2.6	9.1	3.0	9.2	9.1	9.0	3.7	**9.3**	9.2	9.2	35.4
GKD-c_05_n500_m50	500	9.3	7.7	21.0	9.1	2.9	9.1	3.7	9.2	9.1	8.9	5.3	**9.3**	9.2	9.2	42.3
GKD-c_06_n500_m50	500	9.4	7.2	19.0	9.0	2.7	9.0	3.5	9.1	9.0	8.9	4.2	**9.4**	9.3	9.1	87.8
GKD-c_07_n500_m50	500	9.3	7.6	18.0	9.1	2.6	9.1	3.8	9.2	9.1	8.8	4.9	**9.3**	9.3	9.2	44.1
GKD-c_08_n500_m50	500	9.6	7.8	14.0	9.3	2.6	9.3	3.4	9.5	9.4	9.1	6.2	**9.6**	9.5	9.5	30.3
GKD-c_09_n500_m50	500	9.3	7.4	23.0	9.1	3.4	9.1	4.0	9.2	9.1	8.9	5.2	**9.3**	9.2	9.2	44.5
GKD-c_10_n500_m50	500	9.5	7.4	20.0	9.3	2.7	9.3	3.7	9.4	9.3	9.1	5.0	**9.5**	9.4	9.4	46.5
SOM-a_11_n50_m5	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.3	**4.0**	4.0	4.0	0.3
SOM-a_12_n50_m5	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.5	**4.0**	4.0	4.0	0.4
SOM-a_13_n50_m5	50	5.0	**5.0**	0	**5.0**	0	**5.0**	0	**5.0**	5.0	5.0	0.8	**5.0**	5.0	5.0	0.4
SOM-a_14_n50_m5	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.7	**4.0**	4.0	4.0	0.3
SOM-a_15_n50_m5	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.7	**4.0**	4.0	4.0	0.4
SOM-a_16_n50_m15	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.7	**4.0**	4.0	4.0	0.4
SOM-a_17_n50_m15	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.8	**4.0**	4.0	4.0	0.4
SOM-a_18_n50_m15	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.7	**4.0**	4.0	4.0	0.4
SOM-a_19_n50_m15	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.8	**4.0**	4.0	4.0	0.5
SOM-a_20_n50_m15	50	4.0	**4.0**	0	**4.0**	0	**4.0**	0	**4.0**	4.0	4.0	0.7	**4.0**	4.0	4.0	0.5
MDG-b_01_n500_m50	500	-	46.0	45.0	50.6	2.2	50.6	4.4	61.0	52.5	37.7	7.6	**64.6**	61.5	50.8	163.0
MDG-b_02_n500_m50	500	-	35.8	36.0	40.8	2.4	46.0	6.0	52.0	45.9	39.3	7.2	**60.8**	56.3	51.6	204.2
MDG-b_03_n500_m50	500	-	38.0	61.0	41.0	2.4	45.8	5.6	56.3	46.8	39.4	7.8	**60.9**	59.8	54.8	177.7
MDG-b_04_n500_m50	500	-	46.7	113.0	41.4	2.5	47.6	5.6	53.0	44.7	38.1	7.7	**56.6**	54.4	48.8	150.6
MDG-b_05_n500_m50	500	-	47.7	25.0	43.9	2.4	49.1	4.8	54.6	47.3	40.9	7.5	**58.2**	54.0	51.0	210.7
MDG-b_06_n500_m50	500	-	41.3	65.0	41.5	2.5	46.6	5.5	54.1	45.8	33.6	7.5	**60.6**	55.7	49.4	190.6
MDG-b_07_n500_m50	500	-	39.2	84.0	43.9	2.3	47.3	5.1	52.0	45.1	39.6	8.1	**59.3**	54.0	47.9	195.2
MDG-b_08_n500_m50	500	-	36.1	39.0	44.5	2.3	48.7	4.0	55.4	46.6	40.0	7.7	**60.4**	56.2	48.5	185.5
MDG-b_09_n500_m50	500	-	40.4	31.0	43.4	2.3	48.8	6.0	56.0	48.3	38.0	7.6	**60.2**	57.5	52.6	179.0
MDG-b_10_n500_m50	500	-	40.3	148.0	44.1	2.3	50.5	5.2	59.7	47.4	36.8	8.1	**63.0**	61.3	52.7	156.5

*0 means less than 0.01

Table A.1: Detailed results (instance-by-instance results) reported by the proposed SBTS algorithms ($\texttt{SBTS}_{\texttt{short}}$ and $\texttt{SBTS}_{\texttt{long}}$) and the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) on the four sets of 50 benchmark instances with a capacity factor of 0.2.

Instance ($\varphi = 0.3$)	n	f_{opt}	SO (1 run)		SS1 (1 run)		SS2 (1 run)		$\text{SBTS}_{\text{short}}$ (40 runs)				$\text{SBTS}_{\text{long}}$ (40 runs)			
			f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	f_{wst}	$t(s)$	f_{best}	f_{avg}	f_{wst}	$t(s)$
GKD-b_11_n50_m5	50	132.8	**132.8**	0	**132.8**	0.1	**132.8**	0	**132.8**	132.8	132.8	0.8	**132.8**	132.8	132.8	0.2
GKD-b_12_n50_m5	50	159.4	154.7	0	158.1	0	158.1	0	**159.4**	159.4	159.4	0.8	**159.4**	159.4	159.4	0.2
GKD-b_13_n50_m5	50	78.4	77.5	0	**78.4**	0	**78.4**	0	**78.4**	78.4	78.4	0.8	**78.4**	78.4	78.4	0.4
GKD-b_14_n50_m5	50	73.6	71.2	1.0	**73.6**	0	**73.6**	0	**73.6**	73.6	73.6	0.9	**73.6**	73.6	73.6	0.5
GKD-b_15_n50_m5	50	134.0	133.6	0	133.6	0	**134.0**	0	**134.0**	134.0	134.0	1.0	**134.0**	134.0	134.0	0.5
GKD-b_16_n50_m15	50	63.1	**63.1**	0	**63.1**	0	**63.1**	0	**63.1**	63.1	63.1	0.7	**63.1**	63.1	63.1	0.3
GKD-b_17_n50_m15	50	27.9	**27.9**	0	27.5	0	27.5	0	**27.9**	27.9	27.9	0.9	**27.9**	27.9	27.9	0.4
GKD-b_18_n50_m15	50	104.4	**104.4**	0	**104.4**	0	**104.4**	0	**104.4**	104.4	104.4	0.8	**104.4**	104.4	104.4	0.4
GKD-b_19_n50_m15	50	102.5	**102.5**	0	**102.5**	0	**102.5**	0	**102.5**	102.5	102.5	0.8	**102.5**	102.5	102.5	0.4
GKD-b_20_n50_m15	50	101.8	**101.8**	0	**101.8**	0	**101.8**	0	**101.8**	101.8	101.8	0.2	**101.8**	101.8	101.8	0.4
GKD-b_41_n150_m15	150	155.9	151.3	0	154.9	0.2	154.9	0.2	**155.9**	154.2	153.5	3.3	**155.9**	154.7	154.2	16.9
GKD-b_42_n150_m15	150	71.1	67.9	1.0	70.7	0.1	70.7	0.2	**71.1**	71.0	70.7	3.6	**71.1**	71.1	72.2	10.5
GKD-b_43_n150_m15	150	54.3	52.6	2.0	53.9	0.1	53.9	0.2	**54.3**	54.1	54.0	1.8	**54.3**	54.3	54.3	21.1
GKD-b_44_n150_m15	150	90.0	83.1	0	89.2	0.2	87.6	0.2	**90.0**	89.3	89.3	1.2	**90.0**	89.3	89.3	1.1
GKD-b_45_n150_m15	150	97.4	92.7	1.0	95.0	0.2	95.0	0.2	**97.4**	96.5	95.9	3.7	**97.4**	97.1	96.2	15.2
GKD-b_46_n150_m45	150	111.9	105.5	0	110.9	0.2	110.9	0.2	111.6	111.2	110.9	3.8	111.6	111.5	111.2	11.4
GKD-b_47_n150_m45	150	154.8	146.8	2.0	152.9	0.2	152.9	0.2	154.1	154.1	154.1	1.2	154.1	154.1	154.1	0.9
GKD-b_48_n150_m45	150	86.6	82.3	1.0	86.3	0.1	86.3	0.2	**86.6**	86.4	86.2	4.3	**86.6**	86.4	86.3	12.8
GKD-b_49_n150_m45	150	158.5	153.9	0	157.0	0.2	157.0	0.2	**158.5**	158.5	158.5	0.6	**158.5**	158.5	158.5	0.6
GKD-b_50_n150_m45	150	100.6	98.1	1.0	99.5	0.1	99.5	0.1	**100.6**	100.6	100.6	3.1	**100.6**	100.6	100.6	2.1
GKD-c_01_n500_m50	500	8.4	6.8	45.0	8.2	5.3	8.2	5.7	8.3	8.3	8.2	4.1	**8.4**	8.4	8.3	100.6
GKD-c_02_n500_m50	500	8.4	6.3	41.0	8.2	6.0	8.2	6.3	8.3	8.3	8.2	5.2	**8.4**	8.4	8.3	76.3
GKD-c_03_n500_m50	500	8.3	6.5	37.0	8.2	4.7	8.2	5.8	8.2	8.2	8.1	5.0	**8.3**	8.3	8.2	80.2
GKD-c_04_n500_m50	500	8.3	6.5	35.0	8.1	4.8	8.1	5.6	**8.3**	8.2	8.0	6.1	**8.3**	8.3	8.2	89.5
GKD-c_05_n500_m50	500	8.4	6.4	45.0	8.2	4.5	8.2	4.6	8.3	8.2	8.1	6.5	**8.4**	8.3	8.3	17.4
GKD-c_06_n500_m50	500	8.3	6.5	45.0	8.1	5.3	8.1	7.2	**8.3**	8.1	8.1	5.6	**8.3**	8.3	8.2	93.7
GKD-c_07_n500_m50	500	8.4	6.5	43.0	8.2	5.6	8.2	5.9	8.3	8.2	8.1	4.6	8.3	8.3	8.3	24.8
GKD-c_08_n500_m50	500	8.4	6.5	44.0	8.3	5.4	8.3	6.3	**8.4**	8.3	8.3	3.7	**8.4**	8.4	8.4	16.1
GKD-c_09_n500_m50	500	8.3	6.4	42.0	8.2	4.9	8.2	5.3	**8.3**	8.2	8.2	3.8	**8.3**	8.3	8.3	15.8
GKD-c_10_n500_m50	500	8.5	6.5	32.0	8.3	4.5	8.3	5.3	**8.5**	8.3	8.2	5.8	**8.5**	8.5	8.4	66.7
SOM-a_11_n50_m5	50	2.0	**2.0**	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.4	**2.0**	2.0	2.0	0.4
SOM-a_12_n50_m5	50	3.0	1.0	0	2.0	0	2.0	0	**3.0**	2.9	2.0	2.6	**3.0**	3.0	3.0	4.1
SOM-a_13_n50_m5	50	1.0	**1.0**	0	**1.0**	0	**1.0**	0	**1.0**	1.0	0.0	0.3	**1.0**	1.0	1.0	0.4
SOM-a_14_n50_m5	50	2.0	**2.0**	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.4	**2.0**	2.0	2.0	0.4
SOM-a_15_n50_m5	50	2.0	**2.0**	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.6	**2.0**	2.0	2.0	0.8
SOM-a_16_n50_m15	50	2.0	1.0	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.8	**2.0**	2.0	2.0	0.7
SOM-a_17_n50_m15	50	3.0	2.0	0	2.0	0	**3.0**	0	**3.0**	3.0	3.0	0.9	**3.0**	3.0	3.0	1.2
SOM-a_18_n50_m15	50	2.0	1.0	0	**2.0**	0	**2.0**	0	**2.0**	2.0	1.0	2.5	**2.0**	2.0	2.0	3.6
SOM-a_19_n50_m15	50	2.0	**2.0**	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.6	**2.0**	2.0	2.0	0.4
SOM-a_20_n50_m15	50	2.0	1.0	0	**2.0**	0	**2.0**	0	**2.0**	2.0	2.0	0.6	**2.0**	2.0	2.0	0.5
MDG-b_01_n500_m50	500	-	8.2	34.0	20.2	5.0	20.2	7.0	24.5	21.5	17.4	7.2	**27.2**	24.2	19.6	201.7
MDG-b_02_n500_m50	500	-	12.4	47.0	21.1	4.9	22.2	8.7	25.2	22.1	18.8	6.6	**27.6**	25.5	21.5	190.9
MDG-b_03_n500_m50	500	-	7.2	39.0	22.2	5.0	22.2	7.8	25.8	22.8	18.9	6.9	**30.0**	28.0	24.9	208.2
MDG-b_04_n500_m50	500	-	13.4	38.0	22.3	4.8	22.3	7.9	25.9	22.5	17.0	6.8	**28.0**	25.3	23.2	211.0
MDG-b_05_n500_m50	500	-	10.3	39.0	22.9	5.0	22.9	6.8	27.3	23.4	20.1	6.9	**27.9**	25.3	23.0	189.8
MDG-b_06_n500_m50	500	-	10.8	57.0	21.0	4.9	21.0	7.5	26.1	23.1	19.7	7.5	**28.2**	26.0	22.8	223.1
MDG-b_07_n500_m50	500	-	13.3	56.0	22.9	4.9	22.9	8.4	26.1	22.9	19.8	7.6	**28.4**	26.0	20.3	212.6
MDG-b_08_n500_m50	500	-	11.6	55.0	21.6	4.6	22.2	7.4	28.2	23.8	19.9	7.1	**31.1**	28.3	24.2	209.1
MDG-b_09_n500_m50	500	-	13.2	61.0	22.1	4.8	23.2	10.3	25.1	22.2	18.9	6.6	**29.7**	25.9	20.8	208.4
MDG-b_10_n500_m50	500	-	10.5	27.0	21.7	4.8	23.0	8.6	26.4	22.2	18.4	6.7	**29.1**	26.7	21.7	219.8

*0 means less than 0.01

Table A.2: Detailed results (instance-by-instance results) reported by the proposed SBTS algorithms ($\text{SBTS}_{\text{short}}$ and $\text{SBTS}_{\text{long}}$) and the three reference algorithms SO (Peiró et al., 2021), SS1 (Martí et al., 2021b), and SS2 (Martí et al., 2021b) on the four sets of 50 benchmark instances with a capacity factor of 0.3.