Solution-based Tabu Search for the Maximum Min-sum Dispersion Problem

Xiangjing Lai [a], Dong Yue [a] , Jin-Kao Hao [b,c,*], Fred Glover [d]

[a] *Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

[d] *OptTek Systems, Inc., 2241 17th Street Boulder, Colorado 80302, USA*

Information Sciences, 2018
https://doi.org/10.1016/j.ins.2018.02.006

Abstract

The maximum min-sum dispersion problem (Max-Minsum DP) is an important representative of a large class of dispersion problems. Having numerous applications in practice, the NP-hard Max-Minsum DP is however computationally challenging. This paper introduces an effective solution-based tabu search (SBTS) algorithm for solving the Max-Minsum DP approximately. SBTS is characterized by the joint use of hash functions to determine the tabu status of candidate solutions and a parametric constrained swap neighborhood to enhance computational efficiency. Experimental results on 140 benchmark instances commonly used in the literature demonstrate that the proposed algorithm competes favorably with the state-of-the-art algorithms both in terms of solution quality and computational efficiency. In particular, SBTS improves the best-known results for 80 out of the 140 instances, while matching 51 other best-known solutions. We conduct a computational analysis to identify the respective roles of the hash functions and the parametric constrained swap neighborhood.

Keywords: Tabu search; Combinatorial optimization; Dispersion problems; Metaheuristics.

* Corresponding author.
Email addresses: laixiangjing@gmail.com (Xiangjing Lai), medongy@vip.163.com (Dong Yue), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), glover@opttek.com (Fred Glover).

1 Introduction

Given a set $N = \{1, 2, \ldots, n\}$ of n elements and the distance $d_{ij} \geq 0$ $(i < j)$ between each pair of elements, a dispersion problem consists in selecting a set M of elements from N, such that some objective function defined on the selected elements of M is maximized or minimized. In general, according to the objective function, the dispersion problems can be mainly divided into two categories, i.e., efficiency-based dispersion problems and equity-based dispersion problems [1,24,32].

Efficiency-based dispersion problems are only concerned with the dispersion quality of the entire set M. This category of problems mainly includes the maximum diversity problem [2,16,28] and the max–min diversity problem [11,23,25]. The maximum diversity problem (resp. the max–min diversity problem) aims to select a set M with a fixed cardinality m from N, such that the sum of distances (resp. the minimum distance) between the selected elements is maximized. Typical applications of these problems include, for instance, maximally diverse/similar group selection [1], the densest k-subgraph identification [5] and the facility location [14].

Equity-based dispersion problems [24] are to maximize the equity among the selected elements. This category of problems includes the minimum differential dispersion problem (Min-Diff DP) [13,22,32], the maximum mean dispersion problem (MaxMean DP) [6,12,19], and the maximum min-sum dispersion problem (Max-Minsum DP) [1,3,21,24]. The Min-Diff DP aims to minimize the difference between the maximum sum and the minimum sum of the distances from a selected element to the other selected elements; the MaxMean DP tries to maximize the average distance between selected elements, and the Max-Minsum DP requires maximizing the minimum sum of distances between a selected element to the other selected elements. It is worth noting that for the MaxMean DP, the cardinality of M is allowed to vary from 2 to n, while for the other two equity-based dispersion problems, the cardinality of M is fixed to a given positive integer $m < n$. As mentioned in previous studies [1,19,26,32], equity-based dispersion problems have also a number of real-world applications, including urban public facility location [4], equity-based measures in network flow problems [7], selection of homogeneous groups [8], web page ranking [20], community mining [30], etc.

In this work, we focus on the Max-Minsum DP, which is known to be strongly NP-hard [24]. Specifically, the problem can be described as follows. Let $N = \{1, 2, \ldots, n\}$ be a set of n elements, $[d_{ij}]_{n \times n}$ a distance matrix for the given elements, and $m < n$ a fixed positive integer, the Max-Minsum DP problem involves selecting a set $M \subset N$ of cardinality m, such that the minimum sum of the distances between a single selected element and the other selected

elements is maximized. Formally, the Max-Minsum DP problem can be stated as follows:

$$\text{Maximize} \quad Min_{i \in M} \sum_{j \in M} d_{ij} \tag{1}$$

$$\text{Subject to} \quad M \subset N, |M| = m \tag{2}$$

A number of solution approaches exist in the literature for solving the Max-Minsum DP problem. In 2009, Prokopyev et al. proposed a linear mixed integer programming formulation for the Max-Minsum DP and then solved some small instances with $n \leq 100$ using the CPLEX 9.0 solver under a time limit of one hour [24]. At the same time, the authors also devised a GRASP-based metaheuristic and tested its performance on these instances. In 2015, Aringhieri et al. proposed a two-stage metaheuristic method [3]. The proposed method first constructs a high-quality initial solution by removing the less promising components, and then improves the quality of the obtained solution by an attribute-based tabu search method [15]. The algorithm achieved a high performance on the tested instances.

In 2017, Martínez-Gavara et al. designed several GRASP variants coupled with strategic oscillation through constructing the initial solutions by means of a candidate list strategy [21]. The authors also proposed an attribute-based tabu search algorithm to further improve the initial solution generated by the construction procedure. Also in 2017, Amirgaliyeva et al. proposed three variable formulation search (VFS) algorithms that can be viewed as variants of variable neighborhood search (VNS) for solving the Max-Minsum DP, where different but similar optimization objectives are allowed [1]. The computational results on a large number of instances showed that these VFS algorithms are highly efficient compared to previous Max-Minsum DP algorithms. Furthermore, the VFS algorithms improved the best-known results for most instances commonly used in the literature in a short computing time. Consequently, the VFS algorithms can be considered as state-of-the-art algorithms for the Max-Minsum DP.

Compared to the attribute-based tabu search approaches that are very popular [15], the solution-based tabu search approaches [9,10,26,29] have attracted much less attention in the literature. Inspired by a recent study of Wang et al. [26], we introduce in this paper a solution-based tabu search algorithm for the Max-Minsum DP, which proves to be highly effective. The main contributions of this work can be identified as follows:

- We propose the first tabu search algorithm that uses three hash functions to accurately determine the tabu status of neighbor solutions. Compared to the popular attribute-based tabu search methods in the literature, this

solution-based tabu search algorithm has the advantage of not requiring the use of a tabu tenure, which must be tuned.
- Based on a candidate list strategy, we propose a parametric constrained swap neighborhood. With such a constrained neighborhood, the computational efficiency and solution quality of the tabu search algorithm are significantly improved.
- The computational results on six sets of 140 benchmark instances commonly used in the literature show that the proposed algorithm achieves highly competitive performances compared to the state-of-the-art results. Specifically, the proposed algorithm improves the best-known results in terms of the solution quality for 80 out of 140 instances, and matches the best-known results for other 51 instances.

The remainder of the paper is organized as follows. In Section 2, we provide a detailed description of the proposed algorithm. Computational assessment and comparisons based on the extensive experiments are reported in Section 3. In Section 4, we analyze two essential components of the proposed algorithm and shed light on how they affect the performance of the algorithm. In the last section, we provide concluding remarks and perspectives for future work.

2 Solution-based Tabu Search for the Max-Minsum DP

Attribute-based tabu search is a popular metaheuristic approach and has been applied to solve a large number of difficult optimization problems, such as some classic NP-hard combinatorial problems [17,18]. In this approach, one solution attribute or a combination of solution attributes is recorded in a short term memory (tabu list) to prevent the search from revisiting previously encountered solutions. On the other hand, solution-based tabu search is an interesting alternative that records visited solutions (instead of attributes) to avoid search cycling. Viewed more broadly, we may differentiate between fine-grained and coarse-grained solution attributes [15], where customary attribute-based approaches lie toward the coarse-grained end of the spectrum (since they do not closely differentiate among solutions) while solution-based approaches lie at the extreme fine-grained end of the spectrum. Compared to the attribute-based approach, solution-based tabu search is much less studied, mainly due to the high cost of recording whole solutions. In this work, we present an effective solution-based tabu search algorithm for the Max-Minsum DP, which relies on hash vectors to efficiently determine the tabu status of candidate solutions and a constrained swap neighborhood to ensure a high computational efficiency of the algorithm. The use of hash vectors transforms the extreme nature of the solution-based approach into a more moderate version, which nevertheless lies close to the extreme fine-grained end of the spectrum, and, as we show, is highly compatible with the use of an effective candidate list

strategy.

2.1 General Procedure

Algorithm 1: General procedure of the solution-based tabu search algorithm

Input: Instance I, hash vectors H_1, H_2, H_3 with a length of L, hash functions h_1, h_2, h_3, the cutoff time t_{max}

Output: The best solution M^* found

```
1  begin
       /* Initialization of hash vectors, Section 2.5                    */
2      for i ← 0 to L − 1 do
3          H1[i] ← 0
4          H2[i] ← 0
5          H3[i] ← 0
6      end
7      M ← InitialSolution(I)                          /* Sections 2.3 */
8      M* ← M
       /* Main search procedure                                          */
9      while Time() ≤ t_max do
10         Find a best neighbor solution M' that is not simultaneously
           forbidden by the hash vectors (i.e., tabu lists) H1, H2, and H3 from
           the current neighborhood N^c_swap(M)        /* Section 2.4,2.5 */
11         M ← M'                        /* Update the incumbent solution */
12         if f(M) > f(M*) then
13             M* ← M
14         end
           /* Update the hash vectors with M, Section 2.5                */
15         H1[h1(M)] ← 1
16         H2[h2(M)] ← 1
17         H3[h3(M)] ← 1
18     end
19     return M*
20 end
```

To minimize the error rate of determining the tabu status of a candidate solution, SBTS employs three hash vectors associated with three hash functions. The pseudo-code of the proposed algorithm is given in Algorithm 1, where M^* denotes the best solution found so far, H_k $(k = 1, 2, 3)$ represents the hash vectors (i.e., tabu lists), and h_k $(k = 1, 2, 3)$ identifies the corresponding hash functions.

SBTS initializes the hash vectors (lines 2-6) and generates an initial solution M (line 7). Then, the algorithm enters a "while" loop (lines 9-18) to perform

a number of iterations to improve the initial solution. At each iteration, the algorithm first identifies a best non-forbidden neighbor solutions $M^{'}$ from the given neighborhood $N^{c}_{swap}(M)$, and then uses $M^{'}$ to replace the current solution M. Subsequently, the hash vectors (i.e., tabu lists) are updated according to the new solution M (lines 15-17). The loop is repeated until the timeout limit (t_{max}) is reached. Finally, the best solution M^{*} found during the search process is returned as the final result.

2.2 Search Space, Evaluation Function and Solution Representation

Given a Max-Minsum DP instance composed of a set N of n elements, a distance matrix $[d_{ij}]_{n\times n}$ between elements, and a fixed integer m representing the number of selected elements, the search space Ω explored by the SBTS algorithm is composed of all the subsets M of N with a cardinality m, i.e., $\Omega = \{M : M \subset N, |M| = m\}$.

The quality of any candidate solution M in Ω is given by the objective function value $f(M)$:

$$f(M) = Min_{i\in M} \sum_{j\in M} d_{ij} \tag{3}$$

In addition, to facilitate neighborhood operations, we employ an m-dimensional vector S to indicate the set of selected elements and an $(n-m)$-dimensional vector NS to indicate the set of unselected elements. Any solution M in the search space Ω can be represented by these two vectors, i.e., $M =< S, NS >$. Equivalently, a solution can also be represented by a n-dimensional vector $x = (x_1, x_2, \ldots, x_n)$ in which $x_i = 1$ if the element $i \in M$, and $x_i = 0$ otherwise, $i = 1, 2, \ldots, n$.

2.3 Initial Solution

The SBTS algorithm uses a random feasible solution as its initial solution. Specifically, m distinct elements from N are randomly selected to form the initial set M, as shown in Algorithm 2.

2.4 Neighborhood Structure, its Evaluation and Exploration

Like previous studies [1,3,21], our SBTS algorithm is based on the popular swap operator. Given a solution M, an element $u \in M$ and an element $v \in$

Algorithm 2: Initial solution procedure

```
Function InitialSolution()
Input: N = {1, 2, ..., n}, m
Output: A feasible initial solution M
M ← ∅                                  /* M is the set of selected elements */
while |M| < m do
    i ← random() mod n                 /* Randomly pick an element i from N */
    if i ∉ M then
        M ← M ∪ {i}
    end
end
return M
```

$N \setminus M$, the swap operator exchanges u and v to generate a new solution. The full neighborhood induced by the swap operator can be described as follows. Let $< u, v >$ designate a swap move and $M \oplus < u, v >$ be the resulting neighbor solution when applying the move $< u, v >$ to solution M. The full swap neighborhood $N_{swap}^{full}(M)$ is composed of all possible neighbor solutions that can be obtained by applying the swap move to M, i.e., $N_{swap}^{full}(M) = \{M \oplus < u.v >: u \in M, v \in N \setminus M\}$. Clearly, the size of $N_{swap}^{full}(M)$ is exactly equal to $m \times (n - m)$, which becomes very large for relatively large n and m (say several hundreds of elements).

To speed up the tabu search procedure, we devise a parametric constrained swap neighborhood N_{swap}^{c} for the Max-Minsum DP using a candidate list strategy [21,27]. For the parametric constrained swap neighborhood N_{swap}^{c}, the elements to be swapped are limited to two high-quality subsets $X \subset M$ and $Y \subset N \setminus M$. Specifically, given a solution M and two high-quality subsets X and Y, the parametric constrained swap neighborhood $N_{swap}^{c}(M)$ is defined as $N_{swap}^{c}(M) = \{M \oplus < u, v >: u \in X \subset M, v \in Y \subset N \setminus M\}$. However, to construct the N_{swap}^{c} neighborhood, a key issue is the identification of the high-quality subsets X and Y respectively from M and $N \setminus M$.

To identify the subsets X and Y, we maintain an n-dimensional vector $\Delta = (\Delta_1, \Delta_2, \ldots, \Delta_n)$, where $\Delta_i = \sum_{j \in M} d_{ij}$. For the set M, the elements are first sorted by the quick-sort method according to their Δ values in a descending order, and then the first $\rho \times |M|$ elements are chosen as the set X, where ρ is a predetermined parameter. For the set $N \setminus M$, the elements are first sorted by the quick-sort method according to their Δ values in an ascending order, then the first $\rho \times |N \setminus M|$ elements are selected as the set Y. The overall computational complexity of building the constrained neighborhood is $O(mlogm + (n - m)log(n - m))$, which is relatively low. The size of the parametric constrained swap neighborhood is equal to $O(\rho^2 \times m(n - m))$, which is much smaller than that of the full swap neighborhood when the parameter ρ is set to be smaller than 0.5.

Given a neighbor solution M and the corresponding vector $(\Delta_1, \Delta_2, \ldots, \Delta_n)$, the objective value $f(M)$ can be calculated as $f(M) = Min_{i \in M} \Delta_i$ in $O(m)$. Moreover, if two elements $u \in M$ and $v \in N \setminus M$ are exchanged, the vector $\Delta = (\Delta_1, \Delta_2, \ldots, \Delta_n)$ can be rapidly updated in $O(n)$ time as follows:

$$\Delta_i = \begin{cases} \Delta_i - d_{ui}, & \text{for } i = v; \quad (4) \\ \Delta_i + d_{vi}, & \text{for } i = u; \quad (5) \\ \Delta_i - d_{ui} + d_{vi}, & \text{otherwise}; \quad (6) \end{cases}$$

The neighborhood N^c_{swap} is explored by the SBTS algorithm as follows. At each iteration, the SBTS algorithm scans the current whole neighborhood N^c_{swap} and identifies the best candidate solution $M^{'}$ that is not forbidden by the tabu lists from the neighborhood, then replaces the incumbent solution M by the identified solution $M^{'}$ (i.e., $M \leftarrow M^{'}$), as described in Algorithm 1. In the case that all the candidate solutions in N^c_{swap} have been forbidden by the tabu lists, a best candidate solution is selected to replace the incumbent solution, regardless of its tabu status (which is a commonly used aspiration criterion).

2.5 Determination of Tabu Status using Hash Functions

The proposed SBTS algorithm relies on three hash functions and their associated hash vectors to effectively determine the tabu status of candidate solutions. In principle, for a given hash function h and a hash vector H of size of L, h can be used to map a candidate solution $x \in \Omega$ to an index of H ($h : x \in \Omega \rightarrow \{0, 1, 2, \ldots, L-1\}$) such that the binary value of $H[h(x)]$ identifies the tabu status of solution x: $H[h(x)] = 1$ indicates that the solution x has been previously visited and is classified as tabu (thus x is excluded from consideration at the current iteration unless the aspiration criterion is met), while $H[h(x)] = 0$ means that x has not been visited and thus is eligible for consideration at the current iteration.

However, it is well known that collisions may occur with a hash function. That is, two solutions x_1 and x_2 can possibly be mapped to the same position in H, i.e., $h(x_1) = h(x_2)$, leading to a collision. This collision can unfortunately lead to a wrong identification of the tabu status for the concerned candidate solutions (a non-visited solution can be wrongly forbidden to be considered). In order to alleviate this problem, we increase the number of hash vectors as well as the associated hash functions based on two considerations. First, the tabu status of a candidate solution can be conveniently determined by a consolidated rule that *simultaneously* considers the status of the three hash

vectors (as elaborated subsequently in this section). So long as collisions do not *simultaneously* occur in all three hash vectors, there is no misclassification of tabu status in the candidate solutions. Second, the probability that collisions simultaneously occur for $K > 1$ hash vectors significantly decreases as K increases, where K represents the number of hash vectors used. As such, by using three hash functions and hash vectors, we can significantly reduce the probability of misclassifying the tabu status of candidate solutions.

Following the studies in [9,26,29], we adopt the following three hash functions whose hash values can be calculated easily. Let $x = (x_1, x_2, \ldots, x_n)$ be a candidate solution where $x_i = 1$ if element $i \in M$, $x_i = 0$ otherwise. The three hash functions h_k $(k = 1, 2, 3)$ are defined by:

$$h_k(x) = (\sum_{i=1}^{n} \lfloor i^{\gamma_k} \rfloor \times x_i) \ mod \ L \tag{7}$$

where γ_k is a parameter which takes different values for the three hash functions (see Table 1, Section 3.2) and L is the length of the hash vectors, which is set to 10^8 in this work.

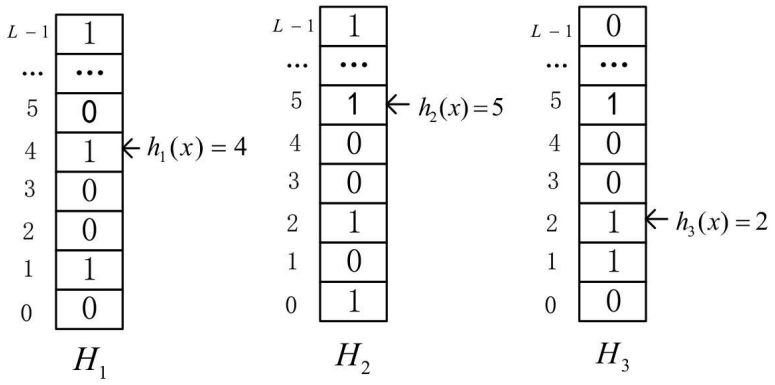


Fig. 1. An illustrative example for determining the tabu status of a given candidate solution x using three hash functions as well as the associated hash vectors.

Now, for a candidate solution x, each hash value $h_k(x)$ $(k = 1, 2, 3)$ indicates the index of x in the associated hash vector H_k. The hash vectors H_k $(k = 1, 2, 3)$ are initialized and updated as follows. At the beginning of the search, the hash vectors H_k $(k = 1, 2, 3)$ are initialized to 0, implying that no solution is forbidden by the tabu list. Then, as the search progresses, the hash vectors H_k $(k = 1, 2, 3)$ are dynamically updated: the values at the indexes $h_k(x)$ of hash vectors H_k $(k = 1, 2, 3)$ are set to 1 once the current solution is replaced by one of its neighbor solutions x, i.e., $H_k[h_k(x)] \leftarrow 1$ $(k = 1, 2, 3)$.

Finally, a key issue is how to determine the tabu status of a candidate solution using these hash vectors and the associated hash functions. At each iteration of our algorithm, for each solution x in the neighborhood (see Section 2.4),

we first check the values of $H_k[h_k(x)]$ $(k = 1, 2, 3)$. If they *all* take the value of 1, the solution x is considered to have been visited previously, and thus is forbidden to be considered again. Otherwise, the solution x is determined as a non-tabu solution. As an illustrative example, Fig. 1 shows a previously visited solution x that is forbidden by the tabu list (because $H_1[h_1(x)]$, $H_2[h_2(x)]$, and $H_3[h_3(x)]$ all take the value of 1). On the other hand, any solution with at least one H_k $(k = 1, 2, 3)$ value equaling 0 is an eligible solution for the current iteration. It is worth mentioning that contrary to the attribute-based tabu strategy, such a tabu strategy makes the notion of tabu tenure irrelevant, thus simplifying the design of the algorithm and reducing the number of required parameters.

Now we consider the computational complexity of calculating the hash values. For a neighbor solution $x \oplus < u, v >$ of the incumbent solution x, where $u \in M$ and $v \in N \setminus M$, the value of $h_k(x \oplus < u, v >)$ can be trivially calculated as $h_k(x \oplus < u, v >) \leftarrow h_k(x) + (\lfloor v^{\gamma_k} \rfloor - \lfloor u^{\gamma_k} \rfloor)$ in $O(1)$, which means that the tabu status of a given neighbor solution can be determined in $O(1)$. Nevertheless, for the initial solution x^o, we need to calculate the corresponding hash values $h_k(x^o)$ $(k = 1, 2, 3)$ from scratch with a complexity of $O(n)$.

2.6 Related Studies

There exist few studies about hash-based tabu search in the literature. In 1993, Woodruff et al. proposed for the first time to use four types of hash functions to record the solutions encountered during recent iterations of the search in a long list [29]. In 1996, Carlton and Barnes made an analysis of these hash functions [9], and devised the first solution-based tabu search approach for the traveling salesman problem with time windows [10]. Recently, Wang et al. proposed a solution-based tabu search approach for the minimum differential dispersion problem using two hash functions [26], which showed an excellent performance on the tested instances.

The SBTS algorithm proposed in this work has a close relation with the above studies. On the one hand, like [9,26], the hash functions used in our SBTS algorithm belong to the classic category of hash functions proposed by Woodruff et al. [29]. On the other hand, there exist some important differences between SBTS and the previous studies. First, our SBTS algorithm is the first solution-based tabu search algorithm designed to exploit the structure of the Max-Minsum DP. Second, SBTS uses three hash vectors and thus has a higher probability of accurately identifying the tabu status of candidate solutions. Third, the SBTS algorithm introduces a new parametric constrained swap neighborhood which plays a key role in the high performance of the proposed algorithm.

3 Experimental Results and Comparisons

In this section, we assess the performance of the proposed algorithm by showing computational results on the commonly used benchmark instances in the literature and by making a comparison with state-of-the-art algorithms.

3.1 Benchmark Instances

The proposed SBTS algorithm was assessed on six sets of 140 benchmark instances [1] . These instances were originally proposed for the maximum diversity problem and subsequently adapted to the Max-Minsum DP in [1,3]. The main characteristics of these instances are as follows.

- APOM Set : This set is composed of 40 instances with $n \in [50, 250]$ and $m \in \{0.2n, 0.4n\}$. For the first 10 instances, the distances d_{ij} between elements are Euclidean, and for the remaining 30 instances the distances d_{ij} are a randomly generated integer number in the interval $[0, 10000]$.
- SOM-b Set : This set is composed of 20 instances with $n \in \{100, 200, 300, 400, 500\}$ and $m \in \{0.1n, 0.2n, 0.3n, 0.4n\}$, where the distances between the elements are a randomly generated integer in the interval $[0, 9]$.
- GDK-c Set : This set is also composed of 20 instances with $n = 500$ and $m = 50$, where the distances d_{ij} between the elements are Euclidean.
- DM1a Set : This set contains 20 instances with $n = 500$ and $m = 200$, where the distances between the elements are a random real number in the interval $[0, 10]$.
- DM1c Set : Like DM1a, this set contains 20 instances with $n = 500$ and $m = 50$, and the distances between the elements are a random real number in the interval $[0, 10]$.
- DM2 Set : This set contains 20 instances with $n = 500$ and $m = 50$, and the distances between the elements are a random real number in the interval $[0, 1000]$.

3.2 Parameter Settings and Experimental Protocol

The proposed SBTS algorithm uses four parameters, whose values were set empirically (see Table 1). The parameter ρ that is used to control the neighborhood size was set to 0.3. The parameters γ_1, γ_2, γ_3 that are used to define

[1] These instances are available at `http://www.di.unito.it/~aringhie/benchmarks.html` and `http://www.optsicom.es/mdp/`

Table 1
Settings of parameters

Parameters	Section	Description	Values
ρ	2.4	Coefficient used in constructing the N^c_{swap} neighborhood	0.3
γ_1	2.5	Parameter used in the first hash function	1.2
γ_2	2.5	Parameter used in the second hash function	1.6
γ_3	2.5	Parameter used in the third hash function	2.0

the three hash functions, were set to 1.2,1.6, and 2.0, respectively. In Section 4, we provide a justification for these choices.

Our SBTS algorithm was programmed in C++ [2] and compiled using the g++ compiler with the -O3 option. All experiments were carried out on a computer with an Intel E5-2670 processor (2.5 GHz and 2G RAM), running the Linux operating system. Following the DIMACS machine benchmark procedure, our machine requires respectively 0.19, 1.17, and 4.54 seconds to solve the graphs r300.5, r400.5, r500.5 [3].

According to the computational results reported in one of the latest papers on the Max-Minsum DP [1] (2017), the attribute-based tabu search method [3] and three variable formulation search (VFS) methods [1] can be considered to be the state-of-the-art algorithms for the Max-Minsum DP. In particular, the VFS methods significantly outperform the GRASP variants proposed in [21]. Moreover, detailed computational results were reported in [1], enabling us to make a direct comparison between our algorithm and these state-of-the-art algorithms. Consequently, in this work we used these four algorithms as our main references to assess the proposed SBTS algorithm.

Due to the stochastic feature of the proposed algorithm, we ran the algorithm 40 times to solve each instance, each run being limited to $t_{max} = n$ seconds, where n is the number of elements in the instance. Note that this time limit was used as the stopping condition in [1].

3.3 Computational Results and Comparison

The computational results of SBTS on the six sets of benchmark instances are summarized in Tables 2–7, together with the results of the four state-of-the-art algorithms, i.e., the tabu search (TS) algorithm [3] and three variable formulation search (VFS) algorithms [1] denoted by VFS2, LS2+VFS1, VFS2+VFS1. Column 1 of each Table gives the instance name (Instance). Column 2 shows the best-known objective values reported in the literature (Best Known) that

[2] The source code of our SBTS algorithm will be available at: `http://www.info.univ-angers.fr/~hao/MaxMinsumdp.html`.

[3] dmclique, `ftp://dimacs.rutgers.edu/pub/dsj/clique`

are compiled from the best results of the reference algorithms. Columns 3–6 list respectively the best results (f_{best}) obtained by the reference algorithms (TS, VFS2, LS2+VFS1, VFS2+VFS1). Specifically, the results of TS in [3] were produced by performing a long run of 50000 iterations, and the results of three VFS algorithms were obtained by running the associated algorithms 10 times on each instance based on a computer with an Intel Core i7 2600 CPU (3.4 GHz and 8 G RAM), where the stopping condition is set to a time limit of n seconds. The computational results of our SBTS algorithm are summarized in columns 7–11, including the best objective values (f_{best}) found over 40 runs, the average objective values (f_{avg}), and the worst objective values (f_{worst}), the standard deviation (σ) of objective values, and the average computing times (t_{avg}) in seconds to reach its final objective value. The improved results in terms of the best objective value (f_{best}) are indicated in bold.

Additionally, in Tables 2 to 7, the rows *#best*, *#equal* and *#worse* indicate respectively the numbers of instances for which the associated algorithm obtains better, equal, or worse objective values in terms of f_{best} compared to the best-known results reported in the literature (Best Known), and the row *avg* denotes the average value over all instances in the set. Finally, to verify whether there exists a significant difference between our SBTS algorithm and the reference algorithms in terms of f_{best}, the *p-values* from the non-parametric Friedman tests are provided in the last row of each table, and a *p-value* smaller than 0.05 implies a significant difference between two compared results.

For this comparative study, we mainly focus on solution quality in terms of the objective values rather than on the computational times. This is because it is difficult to make a fair comparison of computational times due to the differences between the computing platforms, the programming languages, the data structures, and the compilers used by the algorithms. Consequently, although our solution times are quite good, the timing information is given only for indicative purposes.

Table 2 shows that for the APOM instances, our SBTS algorithm outperforms significantly the four reference algorithms. Specifically, SBTS improves the best-known results for 9 out of 40 instances, and matches the best-known results for the remaining 31 instances. Moreover, even the average results of SBTS are better than or match the best-known results for most instances. On the other hand, the computing time to reach the best objective values does not exceed one minute for each instance, and the non-parametric Friedman tests ($p\text{-}values < 0.05$) confirms the significance of these differences. These outcomes indicate that SBTS has a strong search ability and a high computational efficiency on this set of benchmarks. Nevertheless, it should be pointed out that for 4 instances marked by the symbol "*", SBTS with the constrained swap neighborhood of $\rho = 0.3$ failed to find the best-known results and the results in the table were obtained with the full swap neighborhood.

Table 2
Computational results and comparison on APOM instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
01a050m10	95.071	95.071	94.426	95.071	95.071	95.071	95.07	95.071	0.00	0.00
02a050m20	184.495	184.495	184.495	184.495	184.495	184.495	184.50	184.495	0.00	0.00
03a100m20	195.300	195.300	195.300	195.300	195.300	195.300	195.30	195.300	0.00	0.00
04a100m40	370.508	370.508	370.508	370.508	370.508	370.508	370.51	370.508	0.00	0.00
05a150m30	294.565	294.565	294.146	294.565	294.565	294.565	294.57	294.565	0.00	0.02
06a150m60	556.147	556.147	556.147	556.147	556.147	556.147	556.15	556.147	0.00	0.01
07a200m40	392.625	392.625	392.625	392.625	392.625	392.625	392.63	392.625	0.00	0.01
08a200m80	738.100	738.100	737.635	738.100	738.100	738.100	738.10	738.100	0.00	0.03
09a250m50	489.318	489.318	489.310	489.318	489.318	489.318	489.32	489.318	0.00	0.04
10a250m100	921.230	921.230	920.611	921.230	921.230	921.230	921.23	921.230	0.00	0.07
11b050m10	61831	61831	61831	61831	61831	61831	61831.00	61831	0.00	0.00
12b050m20	108715	108248	108715	108715	108715	108715	108715.00	108715	0.00	0.00
13b100m20	118880	118380	118880	118880	118880	118880	118467.50	118380	189.98	0.10
14b100m40	215920	214671	214772	214876	215920	**216176**	216176.00	216176	0.00	0.20
15b150m30	173158	171973	173048	172694	173158	**173400**	173335.60	173178	94.69	0.35
16b150m60	316754	315019	316173	316754	316173	**317541**	317178.98	316865	161.22	2.70
17b200m40	227220	224560	227220	226276	227184	**228451**	228121.63	227924	255.13	3.16
18b200m80	419895	416971	419377	419325	419895	**421092**	420844.13	420299	246.87	6.32
19b250m50	280906	278178	280906	280122	280538	**282850**	282471.95	281929	237.53	10.62
20b250m100	523621	519758	523621	522565	523621	**524752**	524208.48	523768	294.67	52.24
21c050m10	58994	58994	58994	58994	58994	58994	58994.00	58994	0.00	0.00
22c050m20	91338	91338	90568	91338	91338	91338	91338.00	91338	0.00	0.00
23c100m20	112002	112002	112002	112002	112002	112002	112002.00	112002	0.00	0.00
24c100m40	194357	194357	193326	194357	194357	194357	194357.00	194357	0.00	0.00
25c150m30	164357	164357	163479	164357	164357	164357	164357.00	164357	0.00	0.35
26c150m60	290121	290121	287972	290121	290121	290121*	290121.00	290121	0.00	0.44
27c200m40	218548	217583	218222	218548	218313	218548*	218548	218548	0.00	1.30
28c200m80	391851	391587	390322	391851	391851	391851	391851.00	391851	0.00	0.11
29c250m50	272706	270932	271512	272706	271734	272706	272598.03	271847	263.28	4.33
30c250m100	487152	485728	483446	487152	487152	487152*	487152.00	487152	0.00	11.67
31d050m10	74113	74113	74113	74113	74113	74113	74113.00	74113	0.00	0.00
32d050m20	145411	145411	145024	145411	145411	145411	145411.00	145411	0.00	0.00
33d100m20	152236	152236	152190	152236	152236	152236	152236.00	152236	0.00	0.02
34d100m40	295065	294991	295065	295065	295065	295065	295065.00	295065	0.00	0.01
35d150m30	228708	228316	228589	228708	228708	228708	228708.00	228708	0.00	0.02
36d150m60	443707	443413	443277	443707	443707	443707	443650.50	443594	56.50	1.70
37d200m40	304294	303932	304294	304184	304184	304294	304288.50	304184	23.97	1.17
38d200m80	591163	590671	589514	591163	590702	591163*	591163	591163	0.00	8.87
39d250m50	380445	379314	380445	380285	380445	**380770**	380770.00	380770	0.00	1.71
40d250m100	742019	741573	741196	742019	741935	**742429**	742429.00	742429	0.00	1.70
avg	202243.11	201619.88	201808.21	202114.81	202171.93	202431.18	202368.49	202288.56	45.60	2.73
#better	-	0	0	0	0	9				
#equal	-	20	17	32	32	31				
#worse	-	20	23	8	8	0				
p-value	2.7e-3	7.74e-6	2.34e-7	1.57e-3	3.12e-4	-				

Table 3 shows that for all instances in the set GKD-c, our SBTS algorithm achieves the best results (f_{best}) in a short computing time ($\leq$ 10 seconds) with a success rate of 100%. SBTS matches or improves the best results (f_{best}) of three VFS algorithms for all 20 instances. Nevertheless, compared to the tabu search algorithm of [3], SBTS obtains a worse result for 9 out of 20 instances. However, considering the very small differences between our results f_{best} and those reported in [3], it is likely that the differences are caused by round-off errors. Taking the instance GKD-c_17_n500_m50 as an example, the best objective value of the SBTS algorithm is 756.72, and the best result reported in [3] is 756.73. In addition, as in Table 2, for four instances marked by the symbol "*", the results of the SBTS algorithm were obtained with the full swap neighborhood.

Table 3
Computational results and comparison on GKD-c instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
GKD-c_1_n500_m50	762.02	761.98	762.02	762.02	762.02	762.02	762.02	762.02	0.00	3.46
GKD-c_2_n500_m50	771.15	771.15	771.14	771.15	771.14	771.15	771.15	771.15	0.00	0.32
GKD-c_3_n500_m50	764.02	764.02	763.35	764.02	764.02	764.02	764.02	764.02	0.00	9.22
GKD-c_4_n500_m50	763.82	763.81	763.62	763.82	763.82	763.82	763.82	763.82	0.00	0.08
GKD-c_5_n500_m50	765.83	765.63	765.20	765.83	765.60	765.83	765.83	765.83	0.00	1.01
GKD-c_6_n500_m50	761.40	761.34	760.24	761.40	761.40	761.40	761.40	761.40	0.00	0.36
GKD-c_7_n500_m50	764.69	764.68	764.69	764.69	764.69	764.69	764.69	764.69	0.00	0.51
GKD-c_8_n500_m50	766.07	766.07	765.67	766.03	766.03	766.03*	766.03	766.03	0.00	0.45
GKD-c_9_n500_m50	755.16	755.15	755.16	755.16	755.16	755.16	755.16	755.16	0.00	0.06
GKD-c_10_n500_m50	769.00	769.00	768.82	768.96	768.96	768.96	768.96	768.96	0.00	0.84
GKD-c_11_n500_m50	765.74	765.74	764.95	765.71	765.71	765.71	765.71	765.71	0.00	1.29
GKD-c_12_n500_m50	754.50	754.50	754.47	754.47	754.47	754.47	754.47	754.47	0.00	0.06
GKD-c_13_n500_m50	755.97	755.97	755.51	755.94	755.94	755.94*	755.94	755.94	0.00	0.50
GKD-c_14_n500_m50	760.16	760.16	760.16	760.16	760.16	760.16*	760.16	760.16	0.00	0.07
GKD-c_15_n500_m50	757.41	757.41	757.22	757.40	757.40	757.40*	757.40	757.40	0.00	0.12
GKD-c_16_n500_m50	769.81	769.81	769.05	769.80	769.80	769.80	769.80	769.80	0.00	0.77
GKD-c_17_n500_m50	756.73	756.73	756.09	756.72	756.72	756.72	756.72	756.72	0.00	0.89
GKD-c_18_n500_m50	759.67	759.64	759.37	759.67	759.67	759.67	759.67	759.67	0.00	0.93
GKD-c_19_n500_m50	761.52	761.52	760.91	761.52	761.52	761.52	761.52	761.52	0.00	0.64
GKD-c_20_n500_m50	763.97	763.97	763.04	763.94	763.94	763.94	763.94	763.94	0.00	0.46
avg.	762.43	762.41	762.03	762.42	762.41	762.42	762.42	762.42	0.00	1.10
#better	-	0	0	0	0	0				
#equal	-	13	4	11	9	11				
#worse	-	7	16	9	11	9				
p-value	2.7e-3	6.17e-1	1.08e-4	1.0	1.57e-1	-				

From Table 4, which reports the SOM-b instances, our SBTS algorithm improves the best-known results for 13 out of 20 instances, and matches the best-known results for the remaining instances. Furthermore, the worst results (f_{worst}) of the SBTS algorithm are superior to the best-known results for 11 instances, indicating that the SBTS algorithm has a stronger search ability compared to the reference algorithms on this set of benchmarks. In addition, the standard deviations of objective values obtained by the SBTS algorithm are less than 1.0 for all instances, which discloses that SBTS is highly robust. All *p-values* are less than 0.05, indicating there exists a significant difference between the results of the SBTS algorithm and that of each reference algorithm.

Table 5 indicates that for the DM1c instances, the SBTS algorithm improves the best-known results for 18 out of 20 instances, while matching the best-known results for the remaining 2 instances. Furthermore, the average results of SBTS are superior to the best-known results except for 2 instances. The standard deviations of the objective values (σ) do not exceed 0.33 for all instances, again showing that the performance of the SBTS algorithm is strongly robust.

Tables 6 and 7 shows that for the 40 instances in the sets DM1a and DM2, our SBTS algorithm significantly outperforms four reference algorithms. Specifically, SBTS improves the best-known results for all instances without exception. Furthermore, the average results f_{avg} of the SBTS algorithm are also superior to the best-known results for all instances. In addition, the small

Table 4
Computational results and comparison on SOM-b instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
SOM-b_1_n100_m10	62	62	62	62	62	62	62.00	62	0.00	0.01
SOM-b_2_n100_m20	111	110	111	111	111	111	111.00	111	0.00	0.02
SOM-b_3_n100_m30	151	150	151	151	151	151	151.00	151	0.00	0.04
SOM-b_4_n100_m40	194	193	194	194	194	**195**	194.78	194	0.42	0.27
SOM-b_5_n200_m20	117	115	117	117	117	117	117.00	117	0.00	0.10
SOM-b_6_n200_m40	211	207	211	210	211	**212**	211.43	211	0.49	1.93
SOM-b_7_n200_m60	298	293	298	297	298	298	298.00	298	0.00	1.10
SOM-b_8_n200_m80	386	381	386	385	386	**387**	387.00	387	0.00	5.12
SOM-b_9_n300_m30	170	165	170	168	170	170	170.00	170	0.00	2.53
SOM-b_10_n300_m60	308	301	308	306	308	**309**	309.00	309	0.00	9.29
SOM-b_11_n300_m90	440	433	440	439	440	**442**	441.73	441	0.45	41.78
SOM-b_12_n300_m120	571	565	571	571	571	**574**	573.45	573	0.50	36.26
SOM-b_13_n400_m40	222	215	222	219	222	222	222.00	222	0.00	6.90
SOM-b_14_n400_m80	403	396	403	403	403	**407**	406.58	406	0.49	86.16
SOM-b_15_n400_m120	578	570	578	577	578	**583**	582.58	582	0.49	123.21
SOM-b_16_n400_m160	752	743	752	750	750	**757**	756.10	755	0.44	124.11
SOM-b_17_n500_m50	272	262	272	268	271	**273**	273.00	273	0.00	22.07
SOM-b_18_n500_m100	503	492	503	500	502	**505**	504.85	504	0.36	89.14
SOM-b_19_n500_m150	725	713	725	724	725	**729**	728.28	728	0.45	108.12
SOM-b_20_n500_m200	937	926	937	933	936	**942**	940.80	940	0.68	219.34
Avg.	370.55	364.60	370.55	369.25	370.30	**372.30**	372.03	371.70	0.24	43.87
#better	-	0	0	0	0	13				
#equal	-	1	20	8	16	7				
#worse	-	19	0	12	2	0				
p-value	3.12e-4	1.31e-5	3.12e-4	6.33e-5	3.12e-4	-				

Table 5
Computational results and comparison on DM1c instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
01Type1_52.1_n500m50	298.31	291.04	298.31	295.26	298.01	298.31	298.30	298.01	0.07	34.16
02Type1_52.2_n500m50	296.83	289.92	296.83	292.44	295.11	**297.12**	296.98	296.83	0.14	97.64
03Type1_52.3_n500m50	295.70	292.31	295.70	292.80	294.47	**297.37**	296.92	296.56	0.15	65.89
04Type1_52.4_n500m50	295.52	289.66	295.52	291.56	295.52	**296.60**	296.36	296.05	0.09	205.67
05Type1_52.5_n500m50	295.27	289.60	295.27	291.86	295.27	**296.89**	296.89	296.88	0.00	70.74
06Type1_52.6_n500m50	297.69	292.67	297.69	293.39	294.75	**298.26**	298.24	297.33	0.15	183.52
07Type1_52.7_n500m50	295.68	290.15	295.68	294.07	295.11	**296.88**	296.87	296.80	0.02	284.09
08Type1_52.8_n500m50	296.36	290.69	296.36	293.89	296.36	**296.87**	296.71	296.70	0.03	136.88
09Type1_52.9_n500m50	296.52	292.64	296.52	294.09	295.83	**297.87**	297.60	297.40	0.17	91.58
10Type1_52.10_n500m50	298.00	291.78	298.00	294.31	298.00	**298.20**	298.14	297.67	0.13	232.46
11Type1_52.11_n500m50	296.33	291.63	296.33	291.90	295.63	**298.10**	298.10	298.10	0.00	78.41
12Type1_52.12_n500m50	295.13	290.05	295.13	294.23	294.65	**296.44**	296.36	296.35	0.02	100.27
13Type1_52.13_n500m50	296.62	292.82	296.62	294.65	296.42	**299.09**	299.07	298.13	0.15	53.46
14Type1_52.14_n500m50	298.02	291.98	298.02	293.74	296.31	298.02	298.02	298.02	0.00	47.32
15Type1_52.15_n500m50	295.16	291.62	295.16	291.81	294.36	**297.33**	296.37	296.02	0.33	93.69
16Type1_52.16_n500m50	296.95	290.69	296.95	292.81	295.87	**297.99**	297.65	297.60	0.13	67.41
17Type1_52.17_n500m50	294.86	290.90	294.86	293.00	294.86	**297.39**	297.36	297.31	0.04	33.79
18Type1_52.18_n500m50	294.82	289.63	294.82	293.00	294.73	**296.85**	296.78	296.32	0.12	76.79
19Type1_52.19_n500m50	294.14	290.17	294.14	293.85	294.04	**297.17**	296.49	296.47	0.11	181.61
20Type1_52.20_n500m50	295.65	289.86	295.65	290.80	294.96	**296.56**	296.54	296.41	0.05	83.51
avg	296.18	290.99	296.18	293.17	295.51	**297.47**	297.29	297.05	0.09	110.94
#better	-	0	0	0	0	18				
#equal	-	0	20	0	5	2				
#worse	-	20	0	20	15	0				
p-value	2.21e-5	7.74e-6	2.21e-5	7.74e-6	7.74e-6	-				

$p-values$ (≤ 0.05) indicate that there exists a significant difference between the SBTS algorithm and the reference algorithms in terms of f_{best} values on this set of instances.

In summary, the above results show clearly that the proposed SBTS algorithm is very competitive compared to the state-of-the-art algorithms in the

Table 6
Computational results and comparison on DM1a instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
01Type1_52.1_n500m200	1037.24	1034.08	1037.24	1035.98	1036.91	**1043.34**	1042.47	1041.84	0.33	294.48
02Type1_52.2_n500m200	1037.41	1031.22	1037.41	1035.10	1035.63	**1043.15**	1041.75	1040.37	0.67	276.88
03Type1_52.3_n500m200	1034.42	1031.47	1034.21	1033.66	1034.42	**1040.70**	1039.69	1038.60	0.48	295.30
04Type1_52.4_n500m200	1033.59	1029.77	1033.59	1033.43	1033.17	**1041.45**	1039.67	1038.33	0.69	313.38
05Type1_52.5_n500m200	1035.09	1028.80	1035.09	1031.80	1033.73	**1040.46**	1039.33	1037.34	0.72	314.52
06Type1_52.6_n500m200	1035.14	1031.37	1033.91	1034.14	1035.14	**1040.89**	1039.97	1038.54	0.50	277.26
07Type1_52.7_n500m200	1033.43	1029.56	1033.43	1030.83	1033.41	**1039.93**	1038.63	1037.23	0.64	267.42
08Type1_52.8_n500m200	1039.25	1028.46	1039.25	1032.32	1032.19	**1040.35**	1038.99	1037.35	0.82	308.78
09Type1_52.9_n500m200	1034.39	1032.64	1034.39	1034.17	1033.79	**1041.16**	1040.13	1039.12	0.54	262.05
10Type1_52.10_n500m200	1035.56	1030.37	1035.56	1033.75	1034.29	**1040.70**	1039.57	1038.16	0.58	288.89
11Type1_52.11_n500m200	1035.36	1030.54	1035.36	1032.80	1035.16	**1040.27**	1039.15	1037.90	0.54	263.19
12Type1_52.12_n500m200	1033.63	1028.18	1033.63	1031.24	1033.57	**1039.55**	1037.98	1037.08	0.48	315.58
13Type1_52.13_n500m200	1041.66	1036.35	1041.66	1039.58	1040.15	**1047.37**	1046.37	1045.45	0.39	269.24
14Type1_52.14_n500m200	1039.00	1032.48	1037.83	1036.72	1039.00	**1042.70**	1041.91	1040.76	0.44	313.96
15Type1_52.15_n500m200	1037.27	1029.48	1037.27	1033.95	1034.43	**1041.01**	1040.00	1039.18	0.52	318.32
16Type1_52.16_n500m200	1039.92	1033.87	1039.92	1037.37	1039.26	**1045.24**	1044.01	1042.86	0.47	302.91
17Type1_52.17_n500m200	1036.14	1032.40	1036.14	1035.88	1035.89	**1042.97**	1041.88	1040.59	0.52	276.84
18Type1_52.18_n500m200	1035.80	1028.28	1035.80	1031.58	1035.80	**1039.70**	1038.27	1036.62	0.59	297.78
19Type1_52.19_n500m200	1035.07	1030.34	1035.07	1034.24	1033.94	**1040.32**	1039.49	1038.56	0.40	251.92
20Type1_52.20_n500m200	1033.81	1030.55	1033.81	1032.42	1031.59	**1039.61**	1038.03	1036.26	0.67	294.72
avg	1036.16	1031.01	1036.03	1034.05	1035.07	**1041.54**	1040.36	1039.11	0.55	290.17
#better	-	0	0	0	0	20				
#equal	-	0	18	0	4	0				
#worse	-	20	2	20	16	0				
p-value	7.74e-6	7.74e-6	7.74e-6	7.74e-6	7.74e-6	-				

Table 7
Computational results and comparison on DM2 instances

Instance	Best Known	TS	VFS2	LS2+VFS1	VFS2+VFS1	SBTS (this work)				
		f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	f_{worst}	σ	$t_{avg}(s)$
01Type2.1_n500m50	29763.09	29158.15	29763.09	29413.43	29632.54	**29798.64**	29787.19	29777.98	5.27	182.62
02Type2.2_n500m50	29614.05	29073.30	29603.62	29288.74	29614.05	**29773.03**	29773.03	29773.03	0.00	37.51
03Type2.3_n500m50	29656.55	29086.14	29656.55	29639.80	29639.80	**29745.69**	29736.32	29720.14	6.93	187.55
04Type2.4_n500m50	29664.68	29093.40	29664.68	29299.59	29515.35	**29708.41**	29705.71	29659.97	10.12	127.56
05Type2.5_n500m50	29542.69	28924.19	29542.69	29300.63	29495.18	**29683.16**	29677.29	29645.03	8.48	222.82
06Type2.6_n500m50	29554.12	29093.21	29554.12	29321.28	29554.12	**29702.89**	29700.36	29647.56	11.06	124.66
07Type2.7_n500m50	29662.73	29185.38	29662.73	29373.72	29635.87	**29836.19**	29792.80	29750.35	30.31	189.31
08Type2.8_n500m50	29687.32	29156.69	29687.32	29490.62	29631.96	**29828.23**	29772.61	29768.34	10.68	176.92
09Type2.9_n500m50	29563.79	28967.58	29563.79	29235.21	29563.79	**29767.14**	29690.20	29642.92	46.47	226.37
10Type2.10_n500m50	29682.90	28971.87	29682.90	29379.64	29534.88	**29701.99**	29689.10	29638.67	16.63	216.75
11Type2.11_n500m50	29600.94	28984.71	29600.94	29156.46	29600.94	**29697.27**	29675.66	29664.02	15.27	107.77
12Type2.12_n500m50	29644.88	28919.71	29644.88	29193.08	29493.57	**29644.88**	29620.07	29562.82	23.69	82.71
13Type2.13_n500m50	29725.25	29083.33	29725.25	29480.00	29725.25	**29827.41**	29769.41	29751.90	18.67	89.27
14Type2.14_n500m50	29706.63	29218.03	29706.63	29515.23	29677.92	**29920.33**	29898.45	29891.22	5.11	38.36
15Type2.15_n500m50	29746.23	29343.22	29746.23	29507.21	29746.23	**29842.94**	29823.97	29822.43	5.40	43.47
16Type2.16_n500m50	29532.81	29135.51	29523.27	29330.74	29532.81	**29715.71**	29712.97	29688.32	8.22	127.29
17Type2.17_n500m50	29517.15	28993.85	29517.15	29280.40	29517.15	**29789.49**	29712.85	29624.71	58.80	161.73
18Type2.18_n500m50	29584.31	29015.67	29584.31	29133.16	29536.45	**29723.84**	29705.05	29660.17	20.34	162.54
19Type2.19_n500m50	29712.79	29189.34	29688.51	29301.65	29712.79	**29782.17**	29757.19	29736.68	13.68	80.45
20Type2.20_n500m50	29535.90	29132.29	29527.16	29428.72	29535.90	**29726.11**	29690.58	29650.44	13.47	64.91
avg.	29634.94	29086.28	29632.29	29353.47	29594.83	**29760.78**	29734.54	29703.84	16.43	132.53
#better	-	0	0	0	0	20				
#equal	-	0	15	0	10	0				
#worse	-	20	5	20	10	0				
p-value	1.31e-5	7.74e-6	1.31e-5	7.74e-6	7.74e-6	-				

literature both in terms of solution quality and computational efficiency.

4 Analysis and Discussions

The proposed SBTS algorithm includes two essential components, namely the hash functions for determining the tabu status of neighbor solutions and the constrained swap neighborhood. In this section, we turn our attention to an analysis and discussion of these two components.

4.1 Sensitivity Analysis of Hash Functions

Table 8
Influence of the hash functions on the best objective values (f_{best}). Each instance was independently solved 40 times using the SBTS algorithm for each parameter combination in the table, and the best objective values (f_{best}) over 40 runs are respectively reported.

$(\gamma_1, \gamma_2, \gamma_3)$/ Instance	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	avg
(1.1,1.3,1.5)	29798.64	29773.03	29730.45	29708.41	29681.02	29702.89	29836.19	29807.72	29652.99	29682.90	29737.42
(1.1,1.3,1.7)	29790.64	29773.03	29740.76	29708.41	29681.02	29702.89	29772.73	29807.72	29767.14	29701.99	29744.63
(1.1,1.3,1.9)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29836.19	29828.23	29767.14	29701.99	29754.04
(1.2,1.4,1.6)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29772.73	29807.72	29767.14	29701.99	29745.65
(1.2,1.6,1.8)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29828.23	29767.14	29701.99	29754.71
(1.2,1.6,2.0)	29798.64	29773.03	29745.69	29708.41	29683.16	29702.89	29836.19	29828.23	29767.14	29701.99	29754.54
(1.3,1.5,1.7)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29836.19	29770.44	29767.14	29701.99	29748.27
(1.3,1.7,1.9)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29807.72	29767.14	29701.99	29752.66
(1.4,1.6,1.8)	29798.64	29773.03	29745.69	29708.41	29683.16	29702.89	29836.19	29828.23	29767.14	29701.99	29754.54
(1.4,1.8,2.0)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29770.44	29767.14	29701.99	29748.93
(1.5,1.7,1.9)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29807.72	29767.14	29701.99	29752.66
(1.5,1.7,2.0)	29798.64	29773.03	29745.69	29708.41	29683.16	29702.89	29842.85	29807.72	29767.14	29701.99	29753.15
(1.6,1.8,2.0)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29828.23	29767.14	29701.99	29754.71
(1.7,1.8,1.9)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29836.19	29807.72	29767.14	29701.99	29751.99
(1.8,1.9,2.0)	29798.64	29773.03	29740.76	29708.41	29683.16	29702.89	29842.85	29828.23	29767.14	29701.99	29754.71

To check whether the hash functions used by the SBTS algorithm has a significant influence on the performance of algorithm, we carried out an additional experiment based on the first 10 instances of the DM2 set. For the sake of presentation, these instances are renamed as P_1 to P_{10}. Recall that the dimensions of these instances are given by $n = 500$ and $m = 50$. It is relevant to note that, according to the results in Table 7, the results of SBTS have relatively large standard deviations σ in comparison with most instances of other benchmark sets, which implies that these instances are more difficult to solve for the proposed algorithm.

As described in Section 2.5, each parameter γ_k $(k = 1, 2, 3)$ corresponds to a hash function h_k. In this experiment, the values of each γ_k are taken from the set $\{1.1, 1.2, \ldots, 2.0\}$, and 15 representative combinations $(\gamma_1, \gamma_2, \gamma_3)$ of parameters are tested. For each combination of $(\gamma_1, \gamma_2, \gamma_3)$ and instance, the proposed algorithm was run 40 times, and the best objective values (f_{best}) and the average objective values (f_{avg}) are reported in Tables 8 and 9. Column 1

Table 9
Influence of the hash functions on the average objective values (f_{avg}). Each instance was independently solved 40 times using the SBTS algorithm for each parameter combination in the table, and the average objective values (f_{avg}) over 40 runs are respectively reported.

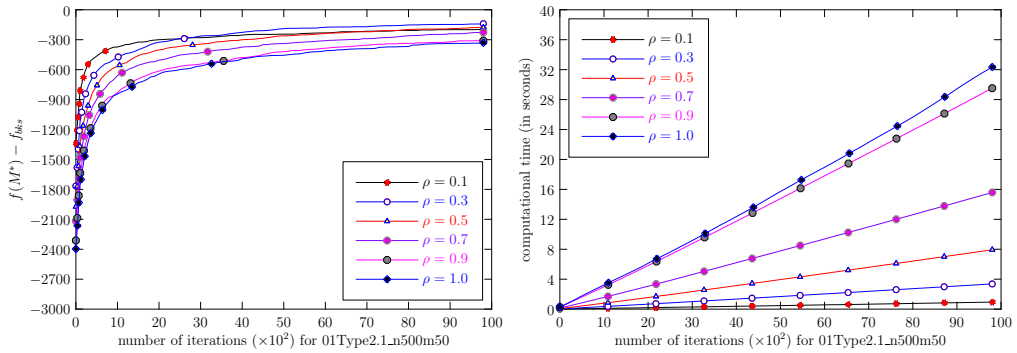
$(\gamma_1, \gamma_2, \gamma_3)$/ Instance	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	avg
(1.1,1.3,1.5)	29745.35	29744.52	29708.37	29653.78	29628.73	29615.01	29745.11	29759.54	29640.88	29650.54	29689.18
(1.1,1.3,1.7)	29764.58	29771.63	29736.41	29677.68	29653.43	29656.69	29759.42	29771.06	29657.02	29680.83	29712.87
(1.1,1.3,1.9)	29784.82	29773.03	29731.13	29700.69	29665.46	29691.54	29775.34	29772.77	29677.58	29689.55	29726.19
(1.2,1.4,1.6)	29769.19	29767.45	29719.47	29705.13	29645.21	29657.51	29744.31	29762.30	29641.13	29660.95	29707.27
(1.2,1.6,1.8)	29784.34	29773.03	29731.51	29695.05	29665.32	29682.66	29772.15	29771.54	29668.50	29687.66	29723.18
(1.2,1.6,2.0)	29787.19	29773.03	29736.32	29705.71	29677.29	29700.36	29792.80	29772.61	29690.20	29689.10	29732.46
(1.3,1.5,1.7)	29767.44	29771.28	29724.19	29697.81	29661.46	29661.48	29773.76	29768.59	29648.73	29680.71	29715.54
(1.3,1.7,1.9)	29783.78	29773.03	29731.64	29707.96	29671.28	29688.95	29784.08	29772.21	29659.86	29692.16	29726.50
(1.4,1.6,1.8)	29779.95	29772.77	29733.66	29699.71	29669.08	29686.98	29774.13	29772.12	29655.63	29697.46	29724.15
(1.4,1.8,2.0)	29788.20	29773.03	29736.82	29708.41	29675.76	29697.49	29781.46	29770.28	29669.78	29692.39	29729.36
(1.5,1.7,1.9)	29789.92	29773.03	29733.64	29708.41	29671.96	29694.62	29786.03	29770.99	29682.91	29691.21	29730.27
(1.5,1.7,2.0)	29789.30	29773.03	29735.09	29707.45	29673.90	29694.83	29787.18	29770.27	29692.47	29695.29	29731.88
(1.6,1.8,2.0)	29789.69	29773.03	29737.10	29707.96	29669.79	29698.64	29783.52	29771.04	29690.96	29693.40	29731.51
(1.7,1.8,1.9)	29795.17	29773.03	29740.26	29705.99	29675.50	29701.75	29780.57	29771.41	29700.26	29693.67	29733.76
(1.8,1.9,2.0)	29789.97	29773.03	29737.89	29708.41	29672.36	29701.16	29782.58	29774.62	29693.33	29684.30	29731.77

and row 1 of the tables respectively indicate the settings of $(\gamma_1, \gamma_2, \gamma_3)$ and the names of instances, and the computational results are reported in columns 2 to 11. The last column shows the average results over the instances tested.

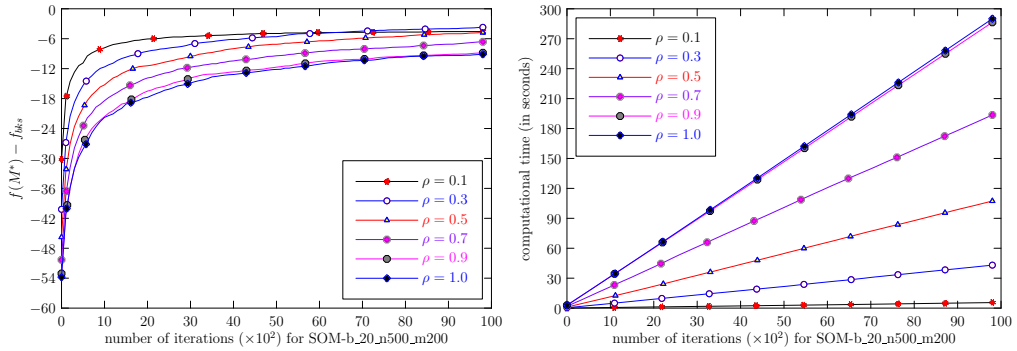
Table 8 indicates that the search ability of SBTS is not very sensitive to the setting of $(\gamma_1, \gamma_2, \gamma_3)$, since the different combinations of parameters led to very similar results in terms of f_{best}. In particular, the table shows that all parameter combinations yield the same best objective value (f_{best}) for 4 out of 10 instances. Moreover, the parameter combinations containing a large value for at least one parameter γ_k are desirable. For example, the combinations $(1.2, 1.6, 2.0)$, $(1.6, 1.8, 2.0)$ and $(1.8, 1.9, 2.0)$ produced the best objective value for 9 out of 10 instances. Furthermore, a similar conclusion can be obtained from Table 9, which shows that in terms of f_{avg} the differences between the results obtained by different parameter combinations are small, and the parameter combinations containing a large value for at least one parameter γ_k yield relatively desirable results in the general case.

4.2 Importance of the Parametric Constrained Swap Neighborhood

The parametric constrained swap neighborhood is another key component of the proposed algorithm, thus in this section we make a detailed analysis of its effect on the algorithm's performance. To this end, we carried out another experiment on two representative instances, namely 01Type2.1_n500m50 and SOM-b_20_n500_m200. For each instance and each value of parameter ρ in the set $\{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$, the proposed SBTS algorithm was independently performed 40 times, each run being given a maximum number of



(a) Evolution of the values of $f(M^*)$ (b) Evolution of running times



(c) Evolution of the values of $f(M^*)$ (d) Evolution of running times

Fig. 2. Comparison between different sizes of the neighborhood

iterations of 10^4. The average results are respectively recorded in terms of the computational times and the gap $(f(M^*) - f_{bks})$ between the best objective value $(f(M^*))$ found so far in the current run (see Algorithm 1) and the best result obtained in this work (f_{bks}). The evolution of computational times and $(f(M^*) - f_{bks})$ as the number of iterations are respectively plotted in Fig. 2. It is worth noting that as illustrated in Section 2.4, the value of parameter ρ directly impacts the size of the constrained swap neighborhood, i.e., $|N^c_{swap}(M)| = \lfloor \rho^2 \times m \times (n-m) \rfloor$, thus a larger value of ρ corresponds to a larger size of neighborhood, and vice-versa.

Fig.2 ((b) and (d)) shows that the computational time increases linearly as the number of iterations, which is consistent with the hypothesis that the computational time of each iteration is proportional to the size of the neighborhood for any value of ρ in (0,1). Moreover, a smaller value of ρ corresponds to a shorter computational time, implying a faster computational speed for each iteration of the SBTS algorithm. On the other hand, the graphs of (a) and (c) of Fig.2 disclose that the algorithm performs better with a smaller value of ρ than with a larger value of ρ, especially at the beginning stage of the algorithm. As the search progresses, the algorithm with a medium sized ρ ($\rho = 0.3$) performs the best. Moreover, the algorithm with a full swap neighborhood (i.e., $\rho = 1.0$) performs the worst during the whole search process.

In summary, for the proposed SBTS algorithm, a small value of ρ that corresponds to a small neighborhood is desirable both in terms of the computational speed and solution quality. Nevertheless, according to the computational results in Section 3.3, the SBTS algorithm with a small neighborhood can occasionally miss the best solutions for some instances, as indicated by the symbol "*" in Tables 2 and 3. For these instances, an adaptive mechanism to adjust the sizes of neighborhood is desirable for reaching a good tradeoff between the computing speed and the search ability.

5 Conclusions and Future work

In this paper, we proposed an effective solution-based tabu search algorithm for the NP-hard maximum min-sum dispersion problem. The key features of the proposed algorithm include a parametric constrained swap neighborhood and a dedicated tabu mechanism based on three hash functions. The constrained swap neighborhood reduces the number of candidate solutions that need to be considered at each search iteration and allows the search procedure to focus on a restricted number of promising candidate solutions. On the other hand, the hash-based tabu mechanism provides the search process with an effective means to avoid cycling and enables the search to continually explore new solutions.

The computational results on six sets of 140 instances commonly used in literature showed that the proposed algorithm is highly effective both in terms of solution speed and solution quality. It is worth noting that the proposed algorithm improves the best-known results for 80 out of 140 instances, while matching the best-known results for other 51 instances. The improved best results (new lower bounds) constitute useful references for evaluating new algorithms for the maximum min-sum dispersion problem.

We also analyzed the impact of the hash functions and the constrained neighborhood on the performance of the algorithm and observed that the algorithm is not sensitive to the hash functions used, but the neighborhood size critically affects the behavior of the algorithm.

Several areas of research invite further investigation. First, our experiments show that the size of the neighborhood has a strong impact on the search efficiency of the algorithm for certain instances. Consequently, it would be interesting to investigate adaptive mechanisms to adjust the neighborhood size during the search process. Second, the proposed algorithm can be further reinforced by exploring additional diversification strategies. To this end, opposition-based search that proved to be successful for the maximum diversity problem [31] is worthy of investigation in the context of the maximum

min-sum dispersion problem. Finally, the ideas of solution-based tabu search and constrained neighborhood are rather general and can be advantageously adapted to solve other dispersion problems as well as similar binary optimization problems.

Acknowledgments

We are grateful to the reviewers for their valuable comments which helped us to improve the paper. This work was partially supported by the National Natural Science Foundation of China (Grant no. 61703213), the Natural Science Foundation of Jiangsu Province of China (Grant no. BK20170904), and Nanjing University of Posts and Telecommunications Science Foundation (NUPTSF) (Grant no. NY217154).

References

[1] Amirgaliyeva Z., Mladenovićb N., Todosijević R., Urošević D., 2017, Solving the maximum min-sum dispersion by alternating formulations of two different problems. *European Journal of Operational Research*, 260, 444-459.

[2] Aringhieri R., Cordone R., 2011, Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society*, 62, 266–280.

[3] Aringhieri R., Cordone R., Grosso A., 2015, Construction and improvement algorithms for dispersion problems. *European Journal of Operational Research*, 242(1), 21–33.

[4] Barbati M., Piccolo C., 2016, Equality measures properties for location problems, *Optimization Letters*, 10(5), 903–920.

[5] Brimberg J., Mladenovićb N., Urošević D., Ngai E., 2009, Variable neighborhood search for the heaviest k-subgraph, *Computers & Operations Research*, 36(11), 2885–2891.

[6] Brimberg J., Mladenovićb N., Todosijević R., Urošević D., 2017, Less is more: Solving the max-mean diversity problem with variable neighborhood search. *Information Sciences*, 382,179–200.

[7] Brown J.R., 1979, The sharing problem, *Operations Research* , 27(2), 324-340.

[8] Brown J.R., 1979, The knapsack sharing problem, *Operations Research*, 27(2), 341–355.

[9] Carlton W.B., Barnes J.W., 1996, A note on hashing functions and tabu search algorithms, *European Journal of Operational Research*, 95(1), 237–239.

[10] Carlton W.B., Barnes J.W., 1996, Solving the traveling salesman problem with time windows using tabu search. *IIE Transactions*, 28, 617–629.

[11] Della Croce F., Grosso A., Locatelli M., 2009, A heuristic approach for the max-min diversity problem based on max-clique. *Computers & Operations Research*, 36(8), 2429–2433.

[12] Della Croce F., Garraffa M., Salassa F.,2016, A hybrid three-phase approach for the max-mean dispersion problem. *Computers & Operations Research*, 71, 16–22.

[13] Duarte A., Sánchez-Oro J., Resende M.G.C., Glover F., Martí R., 2015, Greedy randomized search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, 296, 46–60.

[14] Erkut E., Neuman S., 1989, Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40(3), 275–291.

[15] Glover F., Laguna. M., 1997, Tabu search. *Kluwer Academic Publishers*, Boston.

[16] Glover F., Kuo C.C., Dhir K.S., 1998, Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1), 109–132.

[17] Jin Y., Hao J.K., 2016, Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Information Sciences*, 352, 15–34.

[18] Lai X.J., Hao J.K., Glover F., 2015, Backtracking based iterated tabu search for equitable coloring. *Engineering Applications of Artificial Intelligence*, 46, 269–278.

[19] Lai X.J., Hao J.K., 2016, A tabu search based memetic search algorithm for the max-mean dispersion problem, *Computers & Operations Research*, 72, 118–127.

[20] Kerchove C., Dooren P.V., 2008, The page trust algorithm: how to rank web pages when negative links are allowed? *Proceedings SIAM International Conference on Data Mining*, 346–352.

[21] Martínez-Gavara A., Campos V., Laguna M., Martí R., 2017, Heuristic solution approaches for the maximum minsum dispersion problem. *Journal of Global Optimization*, 67(3), 671–686.

[22] Mladenović N., Todosijević R., Urošević D., 2016, Less is more: Basic variable neighborhood search for minimum differential dispersion problem. *Information Sciences*, 326, 160–171.

[23] Porumbel D.C., Hao J.K., Glover F., 2011, A simple and effective algorithm for the MaxMin diversity problem. *Annals of Operations Research*, 186(1), 275–293.

[24] Prokopyev O.A., Kong N., Martinez-Torres D.L., 2009, The equitable dispersion problem. *European Journal of Operational Research*, 197(1), 59–67.

[25] Resende M.G.C., Martí R., Gallego M., Duarte A., 2010, GRASP and path relinking for the max–min diversity problem. *Computers & Operations Research*, 37(3), 498–508.

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

[27] Wu Q., Hao J.K., 2013, An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization* 26(1), 86–108.

[28] Wu Q., Hao J.K., 2013, A hybrid metaheuristic method for the maximum diversity problem, *European Journal of Operational Research*, 231(2), 452–464.

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

[30] Yang B., W. Cheung, Liu J., 2007, Community mining from signed social networks. *IEEE Transactions on Knowledge & Data Engineering* 19(10), 1333–1348.

[31] Zhou Y., Hao J.K., Duval B., 2017, Opposition-based memetic search for the maximum diversity problem, *IEEE Transactions on Evolutionary Computation*, 21(5): 731–745.

[32] Zhou Y., Hao J.K., 2017, An iterated local search algorithm for the minimum differential dispersion problem, *Knowledge-Based Systems* 125: 26–38.