Tabu Search for the Max-Mean Dispersion Problem

RUBÉN CARRASCO
Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain.
rubensegovia@gmail.com

ANTHANH PHAM TRINH
Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain.
antai.pt@gmail.com

MICAEL GALLEGO
Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain.
micael.gallego@urjc.es

FRANCISCO GORTÁZAR
Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain.
francisco.gortazar@urjc.es

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain.
rafael.marti@uv.es

ABRAHAM DUARTE
Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain.
abraham.duarte@urjc.es

ABSTRACT

In this paper, we address a variant of a classical optimization problem in the context of selecting elements in a set, while maximizing their diversity. In particular, we maximize their mean dispersion. This NP-hard problem was recently introduced as the *maximum mean dispersion problem* (MaxMeanDP), and it models several real problems, from pollution control to capital investment or web page ranking. In this paper, we first review the previous methods for the MaxMeanDP and then explore different tabu search approaches and their influence on the quality of the solutions obtained. As a result, we propose a dynamic tabu algorithm based on three different neighborhoods, which is able to attain high quality solutions. Experimentation on previously reported instances shows that the proposed procedure outperforms existing methods in terms of solution quality. Additionally, we believe that our findings on the use of different memory structures invites further consideration of the interplay between short and long term memory to enhance simple forms of tabu search.

Keywords: Metaheuristics, Tabu Search, Diversity Problems.
Original version: September, 2014

1. Introduction

Diversity problems have been the subject of wide investigation in recent years. In general terms, they consist of maximizing a diversity or dispersion function, which is computed from a subset of selected elements (Glover et al., 1995). Several models (functions) have been proposed in the literature. The most popular is the sum of distances among the selected elements, and results in the well-known Max-Sum Diversity Problem, simply known as the Maximum Diversity Problem (Duarte and Martí, 2007; Gallego et al., 2009). In the Max-Min Diversity Problem, the diversity is computed as the minimum of the distances between each pair of selected elements (Resende et al., 2010). Other models and problems, such as the Maximally Diverse Grouping Problem (Gallego et al., 2013; Rodriguez et al., 2013) deal with the diversity in different subsets.

Prokopyev et al. (2009) proposed four functions and their associated optimization problems to either maximize or balance the diversity among the selected elements: the mean-dispersion function, which maximizes the average dispersion of the selected elements; the generalized mean-dispersion function, which considers vertex-weighted graphs; and the min-sum and the min-diff dispersion functions that consider the extreme equity values of the selected elements. In this paper we focus on Max-Mean Dispersion Problem (MaxMeanDP), which seeks for maximizing the average diversity among selected elements. This strongly NP-hard problem (Prokopyev et al., 2009) has the singularity that the subset of selected elements does not have a fixed pre-established size as in the well-known Max-Sum or Max-Min variants, which makes it especially challenging for heuristic search.

Martí and Sandoya (2013) introduced affinity measures in the MaxMeanDP to generalize the distances among elements by using positive and negative values. These instances are remarkably useful in the context of social networks, where there has been a growing interest in studying social relationships. In particular, these relationships usually model the affinity among elements, where its value can be represented as either an attraction (positive value) or a rejection (negative value) between the elements. The Max-Mean Dispersion Problem has been used to model several real problems, including sentiment analysis, pollution control, capital investment, genetic engineering, web pages ranks, trusting networks, among others (Glover et al., 1998; Wilson et al., 2005; Kerchove and Doore, 2008; Yang et al., 2007).

Given a complete graph $G(V, E)$, where V ($|V| = n$) is the set of elements, and E is the set of edges $\left(|E| = \frac{n(n-1)}{2}\right)$, the MaxMeanDP is formally described as follows. Consider that each edge (i, j) has associated a distance or affinity d_{ij}. A solution $S \subseteq V$ is a subset of any cardinality. Its value, or mean dispersion $md(S)$, is computed as:

$$md(S) = \frac{\sum_{i<j;i,j\in S} d_{ij}}{|S|}$$

Therefore, the Max-Mean problem consists of identifying the set $S^* \subseteq V$ with the maximum mean dispersion value.

Figure 1a shows an example of a graph with 5 vertices and 10 edges and its associated distances. Figures 1b and 1c depict two MaxMeanDP solutions, S_1 and S_2. The selected vertices in each

solution are shown in grey while solid lines highlight their associated edges. The vertices not selected in the solution are shown in white while the edges not in the solution are represented with dashed lines. Notice that the number of selected vertices is different in each solution ($|S_1| = 4$ and $|S_2| = 3$) and the affinities are positive and negative values. To evaluate each solution, we compute its mean dispersion. In particular, Figure 1.b shows a solution where $S_1 = \{1,3,4,5\}$ and $md(S_1) = 4.00$, and Figure 1.c shows the solution $S_2 = \{1,3,4\}$ with $md(S_2) = 4.67$. This example illustrates that the quality of the solutions is not related with their cardinality. Note that in this case, solution S_2 is better than solution S_1, and $|S_2| < |S_1|$.

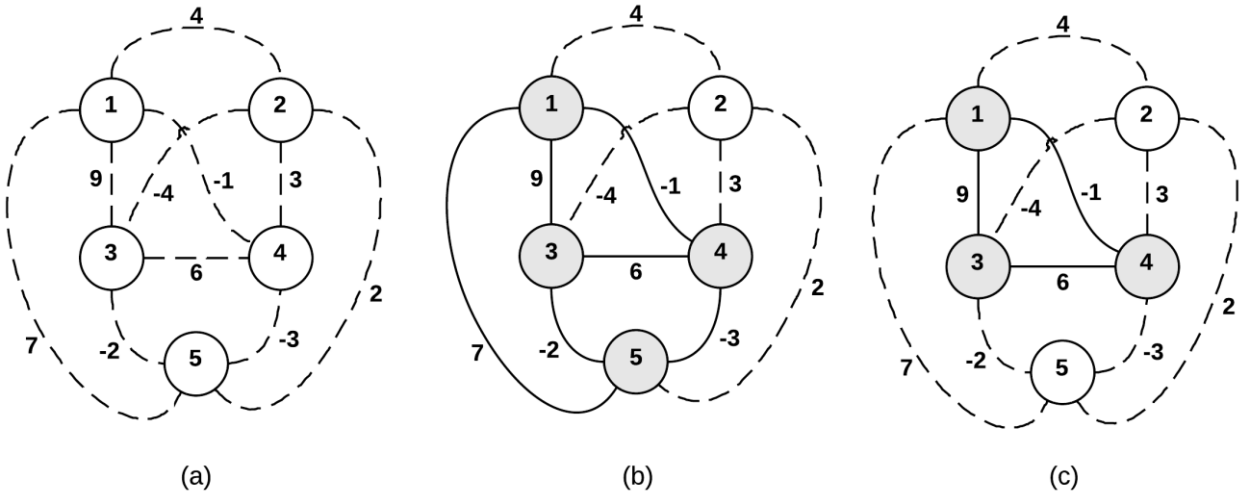

Figure 1. (a) Graph example (b) Solution S_1 (c) Solution S_2.

This problem has recently received some attention. Prokopyev et al. (2009) proposed a linearization of several diversity models, and used Mixed Integer Programming (MIP) to solve instances of moderate size (up to 100 elements) with CPLEX. They also introduced a GRASP method and compared MIP solutions with those obtained with the GRASP procedure in terms of time and optimum gap. The computational experience showed that GRASP obtained high quality solutions in a fraction of time of CPLEX. Martí and Sandoya (2013) introduced a more specialized GRASP with Path Relinking method for the Max-Mean Dispersion Problem. They compared the proposed algorithm with the algorithms described in the related literature. The experimental results showed that the GRASP with Path Relinking method considerable outperformed previous approaches. Therefore, this algorithm can be considered the current state of the art in the context of the Max-Mean Dispersion Problem. The objective of this paper is to propose a specialized Tabu Search procedure to obtain high-quality solutions for the MaxMeanDP.

Tabu search (Glover and Laguna, 1997) is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality. One of the main components of TS is the use of adaptive memory, which creates a flexible search behavior. Memory-based strategies are therefore the key components of tabu search, by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. Memory structures in tabu search are mainly based on four elements: recency, frequency, quality and influence. The first and second elements are related to the short-term and long-term memories respectively,

where each one has its own strategies. Both types of memory have the effect of modifying the neighborhood of a given solution. On the one hand, short-term considerations usually serve to identify characteristics of the neighbor solutions to be excluded. On the other hand, longer-term considerations expand the neighborhood by including promising solutions not ordinarily found on it. In this paper we propose effective mechanism based on the tabu search methodology to find high-quality solutions to the Max-Mean problem.

The rest of the paper is organized as follows: Section 2 describes two new constructive procedures, which are coupled with a local search method based on three neighborhood structures. Short-term and long-term TS variants based on the local search are described in Section 3. In Section 4, we present our exhaustive computational experience. We first analyze and tune the proposed algorithms and then compare our best proposal with the best procedure identified in the literature. Concluding remarks are finally outlined in Section 5.

2. Constructive and local search methods

In this section we propose two greedy algorithms for generating good initial solutions to the Tabu Search procedure. Algorithm 1 shows the pseudo-code of the first greedy procedure called $Const$. It starts by initializing the partial solution to the empty set (step 1) and the Boolean variable which determines the stopping criterion (step 2). $Const$ then iteratively adds vertices to the solution under construction (steps 3 to 11). In order to determine the best candidate to be incorporated in the current partial solution, $Const$ uses the greedy function g (see step 5), which estimates the increment/decrement of the objective function when an element $i \in V \setminus S$ is added to S. The best element is then selected to be added to the partial solution (step 7) or, alternatively, the method detects that it is not possible to further improve S (step 10), returning it in step 12.

PROCEDURE $Const$
1. $S = \emptyset$
2. $notFinished \leftarrow true$
3. **WHILE** $notFinished = true$ **DO**
4. $g(i) \leftarrow \sum_{j \in S} d_{ij}$, for all $i \in V \setminus S$
5. $i^* \leftarrow \arg\max_{j \in V \setminus S} g(j)$
6. **IF** $g(i^*) \geq 0$ **THEN**
7. $S \leftarrow S \cup \{i^*\}$
8. **ELSE**
9. $notFinished \leftarrow false$
10. **END**
11. **END**
12. **RETURN** S

END

Algorithm 1. Pseudo-code of the greedy constructive method $Const$.

The second greedy procedure, $Dest$, proceeds in a similar way (see Algorithm 2) but, instead of adding elements, it removes elements. In particular, it starts by considering all the elements in the solution under construction (step 2). At each iteration, $Dest$ orders the elements in the solution under construction according to g (step 4). Then, instead of selecting the element with

the larger g-vaule, $Dest$ selects the element with the smallest g-value (step 5), removing it from the current solution (step 7). This logic is maintained until no further improvement is possible (step 10).

PROCEDURE $Dest$
1. $S \leftarrow V$
2. $notFinished \leftarrow true$
3. **WHILE** $notFinished \leftarrow true$ **DO**
4. $g(i) = \sum_{j \in S'} d_{ij}, \quad$ for all $i \in S'$
5. $i^* = \arg\min_{j \in S} g(j)$
6. **IF** $g(i^*) \leq 0$ **THEN**
7. $S' = S' \setminus \{i^*\}$
8. **ELSE**
9. $notFinshed \leftarrow false$
10. **END**
11. **END**
12. **RETURN** S'

END

Algorithm 2. Pseudo-code of the greedy constructive method $Dest$.

We additionally propose three different neighborhood structures based respectively on three different types of moves: add, $drop$, and $swap$. Given a solution S, the add-move consists of selecting an element $j \in V \backslash S$ and inserting it in S, producing a new solution S'. For the sake of simplicity we denote $S' = add(S, j)$. Thus, given the solution S, its neighborhood $N_1(S)$ is defined as follows:

$$N_1(S) = \{S' \subseteq V : \ S' = add(S, j) \ , j \in V \backslash S \ \}$$

The second neighborhood is defined with the $drop$-move. It consists of removing an element $i \in S$, producing a new solution S' (i.e., $S' = drop(S, i)$). Then, the second neighborhood of solution S is:

$$N_2(S) = \{S' \subseteq V : \ S' = drop(S, i) \ , i \in S\}$$

We finally propose a composed move, simply called $swap$, which simultaneously removes one element $i \in S$, and adds an element $j \in V \backslash S$, producing a new solution $S' = swap(S, i, j)$. Then, the third neighborhood is formally defined as follows:

$$N_3(S) = \{S' \subseteq V : \ S' = swap(S, i, j) \ , i \in S \text{ and } j \in V \backslash S\}$$

Traditionally, a local search method explores a neighborhood in search for either the first improving or the best improving move. The best solution found at the end of the process is a local optimum (with respect to the neighborhood considered). In this paper, we propose three local search strategies ($\mathrm{LS1}(S)$, $\mathrm{LS2}(S)$, and $\mathrm{LS3}(S)$) based on a nested exploration strategy of the three neighborhoods defined above. Algorithm 3 shows the pseudo-code of the first local search LS1, which for a given solution S, starts by considering the three neighborhoods (step 1). The main loop of LS1 stars by selecting one **neighborhood at random** from the three (step 3), and explores it with a first improvement strategy (step 4). $FirstImpMove$ always returns a

solution S' better than S (if there exists an improving move) or equal to S (no move is finally performed). Then, if S' is better than S, the method updates both the incumbent solution and the composed neighborhood (steps 6 and 7); otherwise, the selected neighborhood is discarder until LS1 finds an improvement move (step 9). LS1 performs iterations until no further improvement is found in the composed neighborhood (steps 2 to 11), returning the best solution found (step 12), which is a local optimum with respect to all neighborhoods.

PROCEDURE $\mathrm{LS1}(S)$
1. $N_{comb}(S) \leftarrow \{N_1(S), N_2(S), N_3(S)\}$
2. **WHILE** $N_{comb}(S) \neq \emptyset$ **DO**
3. $N_x(S) \leftarrow SelectNeighRandom\{N_{comb}(S)\}$
4. $S' \leftarrow FirstImpMove(N_x(S))$
5. **IF** $md(S') > md(S)$ **THEN**
6. $S \leftarrow S'$
7. $N_{comb}(S) \leftarrow \{N_1(S), N_2(S), N_3(S)\}$
8. **ELSE**
9. $N_{comb}(S) \leftarrow N_{comb}(S) \setminus N_x(S)$
10. **END**
11. **END**
12. **RETURN** S

END

Algorithm 3. Pseudo-code of the *LS*1 method.

Algorithm 4 shows the pseudo-code of the second local search LS2. The algorithm starts again by constructing the composed neighborhood (step 1). However, in this procedure, instead of selecting a neighborhood at random, it selects one **move at random** from the set of moves available with the three neighborhoods together (step 3). Then, the procedure tests whether the new solution improves upon the current best solution or not (steps 5 to 10).

PROCEDURE $\mathrm{LS2}(S)$
1. $N_{comb}(S) \leftarrow \{N_1(S), N_2(S), N_3(S)\}$
2. **WHILE** $N_{comb}(S) \neq \emptyset$ **DO**
3. $S' \leftarrow SelectMoveRandom\{N_{comb}(S)\}$
5. **IF** $md(S') > md(S)$ **THEN**
6. $S \leftarrow S'$
7. $N_{comb}(S) \leftarrow \{N_1(S), N_2(S), N_3(S)\}$
8. **ELSE**
9. $N_{comb}(S) \leftarrow N_{comb}(S) \setminus S'$
10. **END**
11. **END**
12. **RETURN** S

END

Algorithm 4. Pseudo-code of the *LS*2 method.

If so, the current best solution is updated as well as the combined neighborhood (steps 6 and 7). Otherwise, the method resorts in the next iteration to another random move selection (in which the discarded move is not considered). This logic is kept until no further improvement is found in the composed neighborhood, returning the local optimum with respect to all neighborhoods (step 12).

```
PROCEDURE LS3(S)
1.      Improved ← true
2.      WHILE Improved = true DO
3.          WHILE Improved = true DO
4.              S' ← BestImpMove(N1(S), N2(s))
5.              IF md(S') > md(S) THEN
6.                  S ← S'
7.              ELSE
8.                  Improved ← false
9.              END
10.         END
11.         S'' ← FirstImpMove(N3(S))
12.         IF md(S'') > md(S) THEN
13              S ← S''
14.             Improved ← true
15.         END
16.     END
17.     RETURN S
END
```

Algorithm 5. Pseudo-code of the *LS*3 method.

The third local search, LS3, performs a more elaborated search strategy. Algorithm 5 shows the pseudo-code of this procedure. It combines a best improvement (steps 3 to 10) with first improvement strategy (steps 11 to 15). In particular, given a solution S, the best improvement part intensively explores the neighborhoods $N_1(S)$ and $N_2(S)$, performing the best available move in both of them together (step 4), with either an add or a $drop$ move. This two neighborhoods are relatively small since $|N_1(S)|+|N_2(S)| = n$. Therefore, this best improvement strategy does not have a large impact on the run time of the method. The final solution of this part is a local optimum with respect to $N_1(S)$ and $N_2(S)$. Then, the local search method explores $N_3(S)$, which is considerably larger than the other two (it has a size of n^2 in the worst case). Therefore, LS3 searches for the first improving move (step 11). If it successes, the incumbent solution is updated (step 13) and the Boolean variable is again set to true (step 14), starting again with the best improvement strategy. Otherwise, the method ends returning the best solution found.

3 Tabu Search

The complete algorithm that we propose for the Max Mean Dispersion Problem consists of a constructive procedure (see Section 2) and a two-phase tabu search method. The first stage (short-term TS) is mainly devoted to the intensification of the search (Section 3.1), and it is based on the local searches described in the previous section. We investigate the effect of a dynamic tabu tenure parameter to manage the short term memory (Section 3.2). The second stage (long-term TS) is focused on a diversification strategy (Section 3.3) to explore new regions of the solution space. Each phase is respectively executed for $maxInt$ and $maxDiver$ iterations. Additionally, both begin from the current solution and after termination they return the overall best solution and their current solution. Note that the current and the best overall are not usually the same solution since these phases usually deteriorate the current solution in order to

escape from its basin of attraction. The search terminates after $maxGlobal$ iterations have elapsed without improving the overall best solution.

3.1. Short-term tabu search

Most of the tabu search designs mainly focus on the short-term memory, where the procedure keeps track of solutions that have changed during the recent past. In order to use this memory, we must select certain attributes that appear in recently visited solutions. These attributes are labeled as tabu-active, which means that solutions that contain them (or combinations of them) become tabu and are excluded from the corresponding neighborhood. In this context, the basic principle of TS is to pursue the local search whenever it encounters a local optimum by allowing non-improving moves. Cycling back to previously visited solutions is prevented by the use of the short-term memory.

Short-term memories are usually implemented as circular lists of fixed length, where the length (referred to as tabu tenure) determines the number of iterations that an attribute is tabu. Note that most of TS designs consider the aspiration criterion, which allows performing a forbidden move if it results in a solution with objective function better than the current best-known solution. The termination criterion is met when the TS performs a determined number of moves without improving the best-known solution. As shown below, in this paper we investigate the use of more complex short-term tabu search approaches, based on several neighborhoods and their associated memory structures.

We propose three short-term tabu search methods based on the three local search algorithms presented in Section 2. We called them TS1 (from LS1), TS2 (from LS2), and TS3 (from LS3) respectively. They consider simultaneously the three neighborhoods ($N_1(S), N_2(S), N_3(S)$), and we define now the associated memory structures. Specifically, given a solution S and its associated neighborhood $N_1(S)$, the reduced neighborhood $N_1^*(S)$ is defined as:

$$N_1^*(S) = \{S' \subseteq V : \ S' = add(S,j) \ , j \in V \backslash \{S \cup T_1\} \ \}$$

where T_1 is a data structure that stores the vertices involved in the corresponding move for *tenure* iterations. Then, as it is customary in tabu search, after a move $add(S,j)$ is done, it is updated by doing $T_1 \leftarrow T_1 \cup \{j\}$, meaning that it is not allowed to perform an add-move involving vertex j during a certain number of iterations. The other two reduced neighborhoods are respectively:

$$N_2^*(S) = \{S' \subseteq V : \ S' = drop(S,i) \ , i \in S \backslash T_2\}$$

$$N_3^*(S) = \{S' \subseteq V : \ S' = swap(S,i,j) \ , i \in S \backslash T_3 \text{ and } j \in V \backslash \{S \cup T_3\}\}$$

where T_2 and T_3 are the associated memory sets. We only describe the adaptation of the short-term TS1 derived from $LS1$ to illustrate how this modification is carried out. Similar adaptations are performed for TS2 and TS3. Basically, we have to change in Algorithm 3 the following elements: (i) the stopping criterion (step 3) is determined by the number of iterations without improvement; (ii) the neighborhoods considered in the step 4 must be the reduced neighborhoods described above; (iii) the move is always performed (step 5) even deteriorating

the value of the objective function; (iv) if the move produces an improvement (steps 6 to 11) the iterations without improvement is set to 0, otherwise, it is incremented; and (v) before executing a new iteration, memories are updated by including the vertex (vertices) involved in the move, and removing those vertices that have been in the memory more than tenure iterations.

3.2. Dynamic tabu tenure

The *tenure* parameter within tabu search determines the number of iterations for which an attribute of a solution (or a set of solutions) is considered tabu. It is nowadays well known that the dynamic modification of this parameter allows the search to react to the recent events, thus making the method more efficient. For example, if the search is currently in a deep and narrow basin of attraction, it is recommended decreasing the tabu tenure to go faster to the local optimum. On the contrary, if the search is in a wide and flat basin of attraction, it is usually better to increase the tabu tenure in order to give more opportunities to escape from that basin of attraction. These strategies have been successfully used in Lü and Hao (2010), Galinier and Hao (1999), and Battitiand and Tecchiolli (1994) to cite a few.

We have identified four different strategies to dynamically adapt the *tenure* parameter (Devarenne et al., 2008). In the first one, called ***time-dependent*** *tenure*, the *tenure* is initialized with a large value, T_{init}, and it is decremented during the search based on either time or on the number of iterations. The process finishes when a minimum value T_{min} is reached. This configuration somehow imitates the behavior of the simulated annealing (Kirkpatrick et al., 1983). Montemanniand and Smith (2001) proposed the expression $T_{new} = \max\{\beta T_{current}, T_{min}\}$ to adjust the *tenure*, where T_{new} is updated after a given number of iterations. Different values of these parameters are studied in the computational experience (see Section 4).

The second strategy, called ***random-bounded*** *tenure*, selects the value of the tenure parameter at random. In general, each move has its own tenure, which means that, after performing it, a random number is generated in the interval $[lb, ub]$, where lb and ub indicates, respectively, the minimum and maximum allowed value for the tenure parameter. The bounds usually depend on some attributes of the problem. The value and effect of these parameters will be studied in the computational experience.

In the third strategy, called ***reactive-tabu*** *tenure,* the value of the *tenure* does not depend on previous history. In particular, it is computed from some attributes of the current solution. We follow the recommendation in Galinier and Hao (1999), who proposed to increase the *tenure* according to the value of some cost function of the current solution S. They specifically proposed to adjust the *tenure* at each iteration with the following equation $T_{new} = L + \lambda F(x)$, where $L \in [0, .., 9]$ is chosen at random, and λ is empirically set to 0.6. F is the cost function which is usually related to the objective function.

Finally, the fourth strategy considered in our method is called ***adaptive-tabu*** *tenure*. In this approach the *tenure* value is adjusted during the search, and the procedure can increase or decrease the value depending on the history and the current solution. Typically the procedure

stats with *tenure* $T = 1$ and it usually maintains a pool Q of the last of last $q = |Q|$ visited solutions. At each iteration, if the current solution is in Q, we have detected a cycle, and the *tenure* is increased according to the equation $T = \min\{\max\{\lfloor 1.1T \rfloor,\ T + 1\}, |V| - 2\}$. Otherwise, the number of iterations without cycles is increased. After a fixed number of iterations without detecting a cycle, we decrement the *tenure* using the expression $T = \max\{\lfloor 0.9T \rfloor, 1\}$. See Devarenne et al. (2008) for a deeper and thorough discussion. The value and effect of these parameters will be studied in the computational experience.

3.3. Long-term Tabu Search

In some applications, the short-term TS memory components are sufficient to produce very high quality solutions. However, in general, TS becomes significantly stronger by including longer-term memory and its associated strategies. In the longer-term TS strategies, the modified neighborhood produced by tabu search may contain solutions not in the original one. Frequency-based memory provides a type of information that complements the information provided by the recency-based memory described in the previous section, broadening the foundation for selecting preferred moves (Glover and Laguna, 1997). We propose two long-term strategies to diversify the search for the MaxMeanDP. These mechanisms are designed to allow the TS to escape from the current basin of attraction.

The diversification stage is executed when the short-term TS reaches the maximum number of iterations without improving the best-found solution. This strategy mainly consists of modifying the values of some attributes of the incumbent solution. We consider two different approaches. The first one, $RandDiver$, consists of performing $maxDiver$ iterations, where each one is a random move (add, $drop$ or $swap$), where the involved vertex (or vertices) is also selected at random. The second strategy, $FreqDiver$, uses the history of the search to guide the process. Specifically, we record the number of times f_i than an element i has appeared in a solution during the whole search. Then, a vertex is probabilistically selected according to frequency count (i.e., f_i for element i). The lower the frequency the larger the probability. The pseudo-code of the diversification stage is shown in Algorithm 7. It stars by initializing the control variables (steps 1 and 2). Then, the procedure modifies the current solution through moves (steps 3 to 23). The vertex involved in the move is determined in step 4. Notice that the selection depends on the specific strategy ($RandDiver$ is a random election, while $FreqDiver$ is a probabilistic election based on the frequency). Then, it is randomly decided in step 5 whether the move is simple ($add/drop$) or complex ($swap$). Then, if the selected vertex is in S or in $V \backslash S$, the diversification method performs one of the four possible moves (steps 6 to 17). The diversification process is abandoned if we found an improvement (steps 18 to 21). Otherwise, it performs iterations until the maximum number of iterations is reached.

As mentioned, our complete tabu search method for the Max Mean Dispersion Problem, from now on simply called TS, consists of two phases, namely short and long term phase, which are alternated. In our computational experimentation shown below, we test the three methods TS1, TS2, and TS3 proposed in this section, to select the best one to constitute the short term phase of TS. We also test, the best dynamic tabu tenure strategy, from the four proposed in this section, to apply to the memory structure in the short term phase. TS terminates after $maxGlobal$ iterations without improving the best solution found.

```
PROCEDURE Diverification(S)
  1.    Improved ← false
  2.    Iters ← 0
  3.    WHILE Improved = false AND iters < MaxDiversify DO
  4.        i ← SelectVertex(V)
  5.        SimpleMove ← {true, false}
  6.        IF i ∈ S AND SimpleMove = true THEN
  7.            S' ← drop(S, i)
  8.        ELSE
  9.            j ← SelectVertex(V\S)
  10.           S' ← swap(S, i, j)
  11.       END
  12.       IF i ∈ V\S AND SimpleMove = true THEN
  13.           S' ← add(S, i)
  14.       ELSE
  15.           j ← SelectVertex(S)
  16.           S' ← Swap(S, i, j)
  17.       END
  18.       IF md(S') > md(S) THEN
  19.           Improved ← true
  20.       END
  21.       S ← S'
  22.       iters ← iters + 1
  23.   END
  24.   RETURN S
END
```

Algorithm 7. Pseudo-code of the *diversification* method.

4. Experimental results

This section describes the computational experiments that we performed to test the efficiency of our Tabu Search procedure as well as to compare it to state-of-the-art methods for solving the MaxMeanDP. We have implemented the TS procedure in Java SE 6 and all the experiments were conducted on an Intel Core 2 Quad CPU and 6 GB RAM.

In our experimentation we consider 40 instances from Martí and Sandoya (2013) divided into two groups: Type I and Type II. Type I instances are symmetric matrices with random numbers generated from the interval $[-10, 10]$. We take 10 instances of size 150 and 10 instances of size 500. Type II instances are symmetric matrices with random numbers from the intervals $[-10, -5] \cup [5,10]$. Again, we take 10 instances of size 150 and 10 instances of size 500. Additionally, using the same random distribution, we generated 20 instances of size 750, 10 from each type, and 20 instances of size 1,000 (10 instances of Type I and 10 instances of Type II). All these 80 instances are available at http://www.optsicom.es/maxmean/.

We have divided our experimentation into two parts: preliminary experimentation and final experimentation. The preliminary experiments were performed to set the values of the key search parameters of our tabu search method as well as to show the merit of its search strategies. We consider a representative subset of instances (10 Type I, 10 Type II, with sizes ranging from 150 to 500).

In our first preliminary experiment we compare the two greedy constructive methods proposed in Section 2. To do that, we generate a solution for each instance, and report for each method, the average percentage deviation (Dev.) with respect to the best known values, the Score statistic (Resende et al., 2010), where the lower the Score, the better the method, and the average CPU time in seconds. Table 1 shows that $Dest$ obtains lower avg. deviation in similar computing times. Additionally, it ranks in the first position (i.e., score equal to 1). We therefore select the former as the constructive method for the rest of our experiments.

	Dev.	**Score**	**Time**
$Const$	11.72%	19	0.03
$Dest$	4.75%	1	0.01

Table 1. Comparison of constructive methods.

In the second preliminary experiment, we assess the quality of the local search methods proposed in Section 2. Each local search is executed after the construction of a single solution. Table 2 shows the associated results over the 20 training instances. In this table, we consider the statistics reported in he previous table and additionally, we consider *#Best*, which indicates the number of times that a method matches the best-known solution.

	Dev.	**#Best**	**Score**	**Time**
$LS1$	2.49%	2	35	0.13
$LS2$	2.20%	4	31	13.11
$LS3$	2.33%	0	37	0.31

Table 2. Comparison of different Local search methods.

Results reported in Table 2 show that $LS2$ obtains the best results in terms of average deviation, Score and number of best solutions found. However, its CPU time is considerably larger than the other two local search methods. We therefore select $LS3$ for the remaining experiments, since it represents a trade-off between quality and computing time.

In our third preliminary experiment we study the contribution of the short-term memory structure (Section 3.2) in a tabu search method. Initially, we set the parameter tenure to 10 in our 3 tabu search variants (we will study the influence of this parameter in the next experiment). The number of iterations without improvement is set to $0.1n$, being n the size of the instance.

	Dev.	**#Best**	**Score**	**Time**
$TS1$	0.52%	13	7	16.47
$TS2$	1.38%	4	27	24.98
$TS3$	0.72%	7	14	15.89

Table 3. Comparison of different tabu search strategies.

Comparing the results reported on Table 3 with those shown in Table 2, we can see that the tabu search methods systematically produce better outcomes than their memory-less counterpart. Specifically, comparing the best method in Table 2 ($LS2$) and the ones reported in Table 3 ($TS1, TS2, TS3$), we observe that the average deviation is decreased in almost two percentage points. Additionally, the number of times that the method matches de best-known solution is considerable improved (from 2 to 13 in the best method). However, as expected, the

computing time is considerable larger. Attending to these results and considering that $TS1$ also presents the lowest score, we select it for the rest of the experimentation.

The fourth preliminary experiment is devoted to compare the four dynamic tenure update strategies described in Section 3.2: time-dependent, random-bounded, reactive-tabu, and adaptive-tabu. We first study the best configuration for each strategy. For the sake of brevity, we do not report here the results of this parameter tuning, and only mention the best configuration found. In particular, for the time-dependent tenure, $t = 0.2n$, $\beta = 0.96$, $its = 100$, and $t_{min} = 10$; for the random-bounded tenure, $\frac{1}{4}n \leq t \leq \frac{1}{2}n$; for the reactive tenure, $t = l + \lambda f(S)$, where l is a random number between 0 and 9, $\lambda = 0.3$, and $f(S)$ is the objective function value of the current solution; and finally, for the adaptive tenure, $t = 1$, with increasing policy $t = 1.1t$ and decreasing policy $t = 0.9t$ (each 20 iterations without a cycle). Once all the parameters are tuned, we compare in Table 4 these 4 variants applied in TS1.

	Dev.	**#Best**	**Score**	**Time**
$TS1$ with time-dependent tenure	0.41%	10	22	54.61
$TS1$ with random-bounded tenure	0.81%	7	50	52.64
$TS1$ with reactive-tabu tenure	0.19%	11	15	54.97
$TS1$ with adaptive-tabu tenure	0.18%	13	15	55.63

Table 4. Performance of different dynamic tenure approaches.

Table 4 shows that the adaptive tenure obtains the best results among the four tested, closely followed by the reactive-tabu strategy. Notice that these strategies increase the computing time, but also increase the quality of the solutions obtained (lower deviation and larger number of best solutions when comparing with the previous experiment). Attending the values of Dev., #Best, and Score reported in Table 5, we conclude that $TS1$ with adaptive-tabu is the best strategy. We complement this experiment by studying the size of the tenure during the time. In particular Figure 2 shows, for a representative instance, the tenure value (Y-axis) and the computing time in seconds (X-axis) for the four strategies. As expected, the random-bounded tenure takes values randomly within the predefined bounds, without any information of the search history. The time-dependent tenure decreases slowly during the search. The reactive-tabu tenure sets the tenure based on the quality of the current solution, thus it increases and decreases with the value of the objective function. Finally, the adaptive-tabu remains almost flat, increasing its value only when no improvement is found after a number of iterations. Then, when the best solution found so far is improved, the value is decreased.

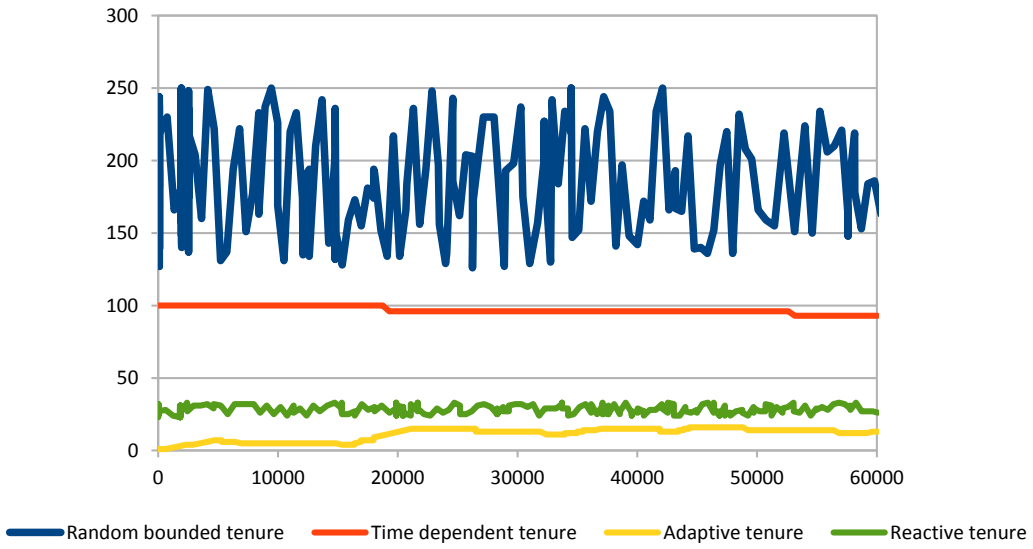

Figure 5. Tenure evolution over time for one instance.

The last preliminary experiment evaluates the contribution of the long-term memories (diversification strategies). In particular, we evaluate the performance of $RandDiver$ and $FreqDiver$ described in Section 3.2. Each strategy is tested with two different values of $maxDiver$ ($0.2n$ and $0.5n$, being n the size of the instance). In order to evaluate the interactions among all the proposed strategies, we execute the complete tabu search method, i.e., $maxInt$ iterations of the short-term tabu search (TS1 with adaptive-tabu tenure) followed by $maxDiver$ iterations of the long-term TS. These two phases are repeated for $maxGlobal$ iterations. Table 5 reports the results for the 4 tested variants. For the sake of simplicity, we denote these variants as TS_MaxMeanDP_1 ($RandDiver = maxDiver = 0.2n$), TS_MaxMeanDP_2 ($RandDiver = maxDiver = 0.5n$), TS_MaxMeanDP_3 ($FreqDiver = maxDiver = 0.2n$), and TS_MaxMeanDP_4 ($FreqDiver = maxDiver = 0.5n$). In order to have a fair comparison, $maxGlobal$ is set to 90 seconds for all variants.

	Dev.	#Best	Score	Time
TS_MaxMeanDP_1	0.45%	7	39	90,16
TS_MaxMeanDP_2	0.25%	14	16	90,14
TS_MaxMeanDP_3	0.07%	14	6	90,13
TS_MaxMeanDP_4	0.26%	14	17	90,22

Table 5. Testing contribution of different diversification strategies.

Table 5 shows relevant information since not all variants are able to outperform the short-term tabu search variants. As a matter of fact, only TS_MaxMeanDP_3 clearly outperforms the results presented in Table 4. Therefore, long-term memories must be carefully designed since straightforward implementations could deteriorate the quality of a short-term tabu search.

In the final experiment, we compare the best-identified tabu search method (TS_MaxMeanDP_3) with state-of-the-art algorithm, i.e., the GRASP with Path Relinking (GRASP_PR) introduced in Martí and Sandoya (2013). We test both methods by using the whole set of 80 instances, divided into 4 subsets of 20 instances with size 150, 500, 750, and 1,000, respectively. The computing time of our method is controlled with the parameter $maxGlobal$.

Considering that the size of the instances varies considerably (from 150 to 1000), we set $maxGlobal$ to 30, 90, 600, and 1800 seconds to instances with size 150, 500, 750, and 1000, respectively. The GRASP_PR method is executed with the parameters recommended in Martí and Sandoya (2013).

Size	Method	Dev.	#Best	Score	Time
150	GRASP_PR	0.25%	3	17	63.07
	TS_MaxMeanDP_3	0.00%	17	3	19.35
500	GRASP_PR	1.45%	2	18	684.11
	TS_MaxMeanDP_3	0.02%	18	2	90.43
750	GRASP_PR	0.90%	3	17	3158.29
	TS_MaxMeanDP_3	0.02%	17	3	601.16
1000	GRASP_PR	1.03%	0	20	10630.90
	TS_MaxMeanDP_3	0.00%	20	0	1803.99

Table 6. Final experiment results.

Table 6 reports the associated results to this experiment. Our method consistently produces better outcomes in all the metrics. In particular, it improves the best-known solution in 72 instances (out of 80). The average deviation across the whole set of instances is 0.01% for TS_MaxMeanDP_3 and 1.00% for GRASP_PR. Additionally, our method is on average more than 5 times faster than the current state-of-the-art algorithm (628 vs. 3634 seconds). We conduct a Wilcoxon test for pairwise comparisons to complement this experiment. This statistical test answers the question: do the two samples (GRASP_PR and TS_MaxMeanDP_3 in our case) represent two different populations? The resulting p-value of 2.61×10^{-4}, 1.9×10^{-5}, 2.67×10^{-5}, and 1.9×10^{-6} on each subset of instances, clearly indicate that the values compared come from different methods (using a typical significance level of $\alpha = 0.05$ as the threshold for reject or not the null hypothesis). Therefore, this statistical test establishes that there are significant differences between both algorithms, confirming the superiority of TS_MaxMeanDP_3 over GRASP_PR.

5. Conclusions

Tabu search has nowadays become the method of choice in numerous metaheuristic implementations and practical applications. In this paper, we have used the context of the Max-Mean Diversity problem to explore some new ideas in this methodology. This problem is of practical significance and, given its complexity, the application of a metaheuristic technology is well justified. The performance of the proposed procedure has been assessed using 80 problem instances of several types and sizes. The procedure has been shown robust in terms of solution quality within a reasonable computational effort. The proposed method is compared with a recently metaheuristic procedure. The comparison favors the proposed tabu search implementation.

An additional important goal of this research has been to investigate the influence of some advance strategies in the context of the tabu search methodology. In particular, we propose a nested exploration of three different neighborhoods, a dynamic modification of memory

structures, and the use of long-term memories for diversification purposes. Our experiments show that a balance between search intensification (exhaustive exploration of local neighborhoods) and diversification (exploration of different regions of the search space), together with flexibility in the search (dynamic variation of the tenure parameter) results in an improved tabu search implementation. However, we have also learnt here an interesting lesson: some advanced tabu search strategies, can lead to poor designs if they are not properly customized. To highlight one of them, we have seen how a long term strategy deteriorates the short term tabu search when coupled with it; while other, better suited for our problem, is able to enrich the method, giving the best design overall.

Acknowledgments

The authors thank Prof. Sandoya for sharing instances and results with them. This research has been partially supported by the Ministerio de Economía y Competitividad of Spain (Grant Ref. TIN2012-35632-C02) and the Generalitat Valenciana (ACOMP/2014/A/241 and Prometeo 2013/049).

References

Battiti, R., G. Tecchiolli (1994) “The reactive Tabu search”. *ORSA Journal on Computing*, 6(2): 126–140.

Battiti, R., I. O. Rayward-Smith, G. Smith (1996) “Reactive Search: Toward Self-Tuning Heuristics”, 61–83. John Wiley and Sons Ltd.

Devarenne, I., H. Mabed, A. Caminada (2008) “Adaptive Tabu Tenure Computation in Local Search”. *Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science, 4972:1-12.

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

Duarte, A., R. Martí, F. Glover, F. Gortázar (2011) “Hybrid scatter tabu search for unconstrained global optimization”. *Annals of Operations Research*; 183(1): 95-123.

Galinier, P., J. K. Hao (1999) “Hybrid Evolutionary Algorithms for Graph Coloring”. *Journal of Combinatorial Optimization*, 3: 379–397.

Gallego, M., M. Laguna, R. Martí, A. Duarte (2013) “Tabu search with strategic oscillation for the maximally diverse grouping problem”. *Journal of the Operational Research Society*, 64: 724–734.

Gallego, M., A. Duarte, M. Laguna, R. Martí (2009) “Hybrid heuristics for the maximum diversity problem”. *Computational Optimization and Applications*, 44(3): 411-426.

Glover, F., M. Laguna (1997). *Tabu search*. Kluwer Academic Publishers.

Glover F, Kuo CC, Dhir KS (1995) “A discrete optimization model for preserving biological diversity”. *Applied Mathematical Modeling*, 19: 696–701.

Glover, F., CC. Kuo, KS. Dhir (1998) “Heuristic algorithms for the maximum diversity problem”. *Journal of Information and Optimization Sciences*, 19(1): 109–132.

Kerchove, C.,P. V. Dooren (2008) “The Page Trust algorithm: how to rank web pages when negative linksare allowed?”. In *Proceedings SIAM International Conference on Data Mining*, 346–352.

Kirkpatrick, S., C.D. Gelatt, M.P. Vecchi (1983). "Optimization by Simulated Annealing". *Science* 220 (4598): 671–680.

Lozano, M., A. Duarte, F. Gortázar, R. Martí (2013) “A hybrid metaheuristic for the cyclic antibandwidth problem”. *Knowledge Based Systems*, 50: 103-113.

Lü, Z. P., J. K. Hao (2010), “Adaptive tabu search for course timetabling”. *European Journal of Operational Research*; 200 (1): 235-244.

Martí, R., F. Sandoya (2013) “GRASP and Path Relinking for the Equitable Dispersion Problem”. *Computers and Operations Research*; 40(12): 3091-3099.

Montemanni, R., D.H., Smith (2001) “A Tabu search Algorithm with a dynamic Tabu list for the Frequency Assignment Problem”. *Technical Report*, University of Glamorgan.

Prokopyev, O.A., N. Kong, DL. Martinez-Torres (2009) “The equitable dispersion problem”. *European Journal of Operational Research*, 197:59–67.

Resende, M., R. Martí, M. Gallego, A. Duarte (2010) “GRASP and path relinking for the max-min diversity problem“. *Computers and Operations Research*, 37:498–508.

Rodríguez, F. J., M. Lozano, C. García-Martínez, J. D. González-Barrera (2013) “An artificial bee colony algorithm for the maximally diverse grouping problem”. *Information Sciences*; 230:183-196.

Wilson, T., J. Wiebe, P. Hoffmann (2005) “Recognizing contextual polarity in phrase-level sentiment analysis”. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 347-354.

Yang, B., W. Cheung, J. Liu (2007) “Community mining from signed social networks”. *IEEE Transactions on Knowledge and Data Engineering*, 19(10):1333–1348.