Knowledge-Based Systems 85 (2015) 256–264

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Tabu search for the Max–Mean Dispersion Problem

Rubén Carrasco [a], Anthanh Pham [a], Micael Gallego [a], Francisco Gortázar [a], Rafael Martí [b], Abraham Duarte [a,*]

[a] Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain
[b] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain

ARTICLE INFO

Article history:
Received 22 September 2014
Received in revised form 5 May 2015
Accepted 9 May 2015
Available online 15 May 2015

Keywords:
Optimization
Metaheuristics
Tabu search
Diversity problems

ABSTRACT

In this paper, we address a variant of a classical optimization model in the context of maximizing the diversity of a set of elements. In particular, we propose heuristics to maximize the mean dispersion of the selected elements in a given set. This NP-hard problem was recently introduced as the *maximum mean dispersion problem* (MaxMeanDP), and it models several real problems, from pollution control to ranking of web pages. 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 search algorithm, based on three different neighborhoods. Experiments on previously reported instances show that the proposed procedure outperforms existing methods in terms of solution quality. It must be noted that our findings on the use of different memory structures invite to further consideration of the interplay between short and long term memory to enhance simple forms of tabu search.

1. Introduction

Diversity problems have been recently the subject of investigation in the area of optimization. In general terms, they consist of maximizing a diversity or dispersion function on a subset of selected elements [10]. Several models have been proposed in the literature. The most popular is given by 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 [3,8]. On the other hand, in the Max–Min Diversity Problem, the diversity is computed as the minimum of the distances between each pair of selected elements [20]. Special attention deserves the Maximally Diverse Grouping Problem [7,22], which deals with the diversity in different subsets, and the Max–Mean Dispersion Problem (MaxMeanDP), which seeks for maximizing the average diversity among selected elements. This paper focuses on this latter variant, introduced in Prokopyev et al. [19], for which Martí and Sandoya proposed a GRASP with Path Relinking method [17].

The MaxMeanDP is strongly NP-hard [19], and has the singularity that the subset of selected elements does not have a fixed pre-established size, as it happens in the well-known Max–Sum or Max–Min variants. This characteristic makes this variant especially challenging for heuristic search. The MaxMeanDP has been used to model several real problems, including sentiment analysis, pollution control, capital investment, genetic engineering, web pages ranking, and trusting networks among others [11,13,23,24].

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 formulated as follows. Consider that each edge $(i,j) \in E$ has an associated 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:

$$md(S) = \frac{\sum_{i<j;i,j\in S} d_{ij}}{|S|} \qquad (1)$$

where $|S|$ is the number of elements in S. Therefore, the MaxMeanDP problem consists of identifying the set $S^* \subseteq V$ with the maximum mean dispersion value.

The main objective of this paper is to propose an efficient procedure, based on the Tabu Search methodology [9], to obtain high-quality solutions for the MaxMeanDP. The contributions of our work can be summarized as follows:

* Corresponding author.
E-mail addresses: rubensegovia@gmail.com (R. Carrasco), antai.pt@gmail.com (A. Pham), micael.gallego@urjc.es (M. Gallego), francisco.gortazar@urjc.es (F. Gortázar), rafael.marti@uv.es (R. Martí), abraham.duarte@urjc.es (A. Duarte).

http://dx.doi.org/10.1016/j.knosys.2015.05.011
0950-7051/© 2015 Elsevier B.V. All rights reserved.

- Design and implementation of two greedy procedures based on constructive and destructive moves.
- Design and implementation of three local search methods based on a nested exploration of different neighborhood structures.
- Experimental study of a Tabu Search algorithm, with short-term and long-term strategies.
- Comparison of new and existing methods for the MaxMeanDP.

We start by exploring the existing MaxMeanDP literature in Section 2. Then, Section 3 describes two new greedy procedures, which are coupled with a local search method based on three neighborhood structures. Short-term and long-term TS variants based on the previously described local search methods are described in Section 4. In Section 5, we present our exhaustive computational experience. In particular, 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 6.

2. Previous methods on the Max–Mean diversity problem

As mentioned in Section 1, diversity models have recently received much attention. Prokopyev et al. [19] proposed four models and their associated optimization problems to either maximize or balance the diversity among the selected elements: the mean-dispersion model, which maximizes the average dispersion of the selected elements; the generalized mean-dispersion model, which considers vertex-weighted graphs; and the Mim–Sum and the Min–Diff dispersion models that consider the extreme equity values of the selected elements. In this paper we focus on the first one, which leads to the Max–Mean Dispersion Problem (MaxMeanDP).

Prokopyev et al. [19] proposed a linearization of several diversity models, including the MaxMeanDP, and used Mixed Integer Programming (MIP) to solve instances of moderate size (up to 100 elements) with CPLEX. They also introduced a GRASP method [5] 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 the time employed by CPLEX.

Martí and Sandoya [17] introduced a more specialized GRASP with Path Relinking method for the MaxMeanDP. They compared the proposed algorithm with the procedures described in the related literature. The experimental results showed that the GRASP with Path Relinking method considerably outperformed previous approaches. Therefore, this algorithm can be considered as the current state of the art in the context of the Max–Mean Dispersion Problem.

Martí and Sandoya [17] also introduced affinity measures in the context of 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. We will consider these instances in our computational comparison.

For a better understanding of the MaxMeanDP, we introduce in Fig. 1 an illustrative example. In particular, Fig. 1a shows a graph with 5 vertices and 10 edges and its associated distances or affinities (taking both positive and negative values). Fig. 1b and 1c depict two MaxMeanDP solutions, S_1 and S_2 respectively, for the graph shown in Fig. 1a. The selected vertices are shown in gray in both solutions, while solid lines highlight their associated edges. The vertices not selected are shown in white, while the edges not in the solution are represented with dashed lines. Note that a solution for MaxMeanDP can have any number of selected vertices between 2 and the number of vertices in the graph, that is, $|S| \in [2, |V|]$. In our example, $|S_1| = 4$ and $|S_2| = 3$.

To evaluate each solution, we compute its mean dispersion. It is calculated as the sum of the distances between the selected vertices divided by the number of selected vertices. In particular, Fig. 1b shows a solution where $S_1 = \{1, 3, 4, 5\}$ and $md(S_1) = (9 - 1 - 3 + 6 - 2 + 7)/4 = 16/4 = 4.00$, and Fig. 1c shows the solution $S_2 = \{1, 3, 4\}$ with $md(S_2) = (9 - 1 + 6) = 14/3 = 4.67$. This example illustrates that the quality of the solutions is not related to their cardinality. Note that in this case, solution S_2 is better than solution S_1 (MaxMeanDP is a maximization problem), although $|S_2| < |S_1|$.

3. Heuristic methods

In Section 2, we have identified two previous methods for the MaxMeanDP based on the GRASP methodology [5]. This

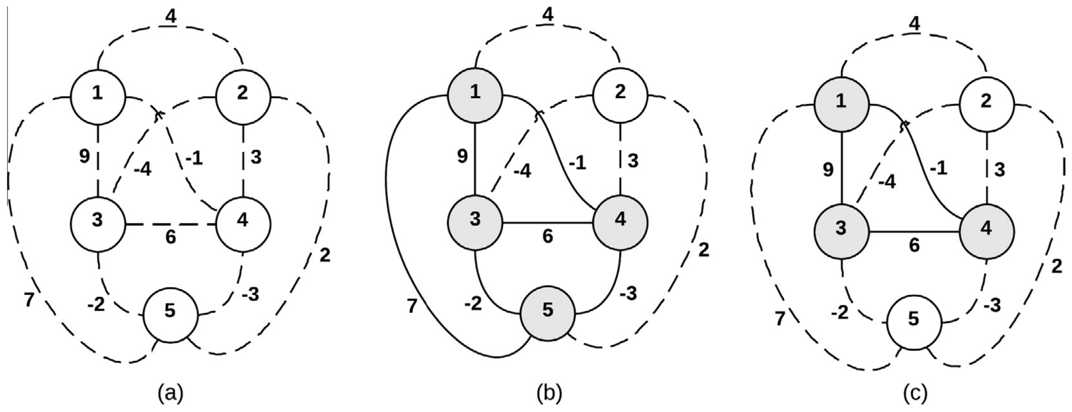

Fig. 1. (a) Graph example. (b) Solution S_1. (c) Solution S_2.

methodology basically relies on the interplay between greediness and randomization and ignores memory structures in its design. It is however well document [9], that the implementation of memory structures often produces high-quality outcomes in hard combinatorial optimization problems, such as the one considered here. We therefore undertake to explore the adaptation of the tabu search methodology, the hallmark of memory-based designs, to the MaxMeanDP. In this section we define the foundations of our tabu search algorithm. Specifically, we first propose two greedy algorithms for generating high-quality initial solutions to start the search of our procedure. Then, we propose three different neighborhood structures and a local search based on them. In the next section we will show the addition of memory structures to these elements to obtain our final tabu search method.

3.1. Obtaining the initial solution

In this subsection we propose a constructive and a destructive strategy to obtain an initial solution. The former works by selecting elements (i.e., adding elements one by one to a partial solution S, which is initially empty), as long as the mean dispersion of the selected elements increases. The latter considers that S initially contains all the elements in the graph (i.e., all of them are selected), and removes elements from S (one by one) as long as the mean dispersion increases.

Algorithm 1 shows the pseudo-code of the first procedure called *Const*. It starts by initializing the partial solution S 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 3–11). 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.

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

```
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 $g(i^*) \geqslant 0$ THEN
7.      $S \leftarrow S \cup \{i^*\}$
8.    ELSE
9.      $notFinished \leftarrow false$
10.   END
11. END
12. RETURN $S$
END
```

The second procedure, *Dest*, proceeds in a similar way (see Algorithm 2), In this case, instead of adding elements, it removes elements from S.

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

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

In particular, it starts by considering all the elements in the solution S under construction (step 1). 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 largest g-value, *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).

3.2. Improving solutions with local search

In this subsection we propose a local search method based on three moves: *add*, *drop*, and *swap*. As it is customary in a local search method, it improves a given solution by iteratively performing a small change called *move*. In particular, we consider the initial solution S, obtained with one of the two greedy methods introduced above and, at each iteration, we explore the set of solutions that can be obtained by performing a move (usually called the *neighborhood*). If any of them improves upon the current solution, we perform the move; otherwise the local search finishes.

Given a solution S, the *add*-move consists of selecting an element $j \in V \setminus S$ 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:

$$N_1(S) = \{S' \subseteq V : S' = add(S, j), \quad j \in V \setminus S\} \tag{2}$$

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:

$$N_2(S) = \{S' \subseteq V : S' = drop(S, i), i \in S\} \tag{3}$$

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

$$N_3(S) = \{S' \subseteq V : S' = swap(S, i, j), i \in S \text{ and } j \in V \setminus S\} \tag{4}$$

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 (LS1(S), LS2(S), and 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 starts 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 discarded until LS1 finds an improvement move (step 9). LS1 performs iterations until no further improvement is found in the composed neighborhood (steps 2–11), returning the best solution found (step 12), which is a local optimum with respect to all neighborhoods.

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

```
PROCEDURE 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 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 5–10). 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 no longer 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).

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

```
PROCEDURE 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
```

The third local search, LS3, performs a more elaborated search strategy. Algorithm 5 shows the pseudo-code of this procedure. It combines best improvement (steps 3–10) and first improvement strategies (steps 11–15). 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. These 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 succeeds, 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.

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

```
PROCEDURE LS3(S)
1.  $Improved \leftarrow true$
2.  WHILE $Improved = true$ DO
3.    WHILE $Improved = true$ DO
4.      $S' \leftarrow BestImpMove(N_1(S), N_2(s))$
5.      IF $md(S') > md(S)$ THEN
6.        $S \leftarrow S'$
7.      ELSE
8.        $Improved \leftarrow false$
9.      END
10.   END
11.   $S' \leftarrow FirstImpMove(N_3(S))$
12.   IF $md(S'') > md(S)$ THEN
13.     $S \leftarrow S''$
14.     $Improved \leftarrow true$
15.   END
16. END
17. RETURN S
END
```

4. Tabu search

Tabu search [9] 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 [4,15].

Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each solution S has an associated neighborhood $N(S)$, and each solution $S' \in N(S)$ is reached from S by an operation called a move. We may contrast TS with a simple descent local search method. Such a method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found, thus obtaining a local optimum, which in most cases will not be a global optimum. TS permits moves that deteriorate the current objective function value but the moves are chosen from a modified neighborhood $N^*(S)$. Short and long term memory structures are responsible for the specific composition of $N^*(S)$. In other words, the modified neighborhood is the result of maintaining a selective history of the states encountered during the search.

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. 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 mechanisms based on the tabu search methodology to find high-quality solutions to the MaxMeanDP.

The complete algorithm that we propose for the Max–Mean Dispersion Problem consists of a constructive procedure (see Section 3) and a two-phase tabu search method. The first stage (short-term TS) is mainly devoted to the intensification of the search (Section 4.1), and it is based on the local search methods described in the previous section. We investigate the effect of a dynamic tabu tenure parameter to manage the short-term memory (Section 4.2). The second stage (long-term TS) is focused on a diversification strategy (Section 4.3) 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.

4.1. 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:

$$N_1^*(S) = \{S' \subseteq V : S' = add(S,j), \quad j \in V \setminus \{S \cup T_1\}\} \tag{5}$$

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:

$$N_2^*(S) = \{S' \subseteq V : S' = drop(S,i), i \in S \setminus T_2\} \tag{6}$$

$$N_3^*(S) = \{S' \subseteq V : S' = swap(S,i,j), i \in S \setminus T_3 \text{ and } j \in V \setminus \{S \cup T_3\}\} \tag{7}$$

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 6–11) the parameter "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.

4.2. 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 [16], Galinier and Hao [6], and Battiti and Tecchiolli [1] to cite a few.

We have identified four different strategies to dynamically adapt the *tenure* parameter [2]. 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 [14]. Montemanni and Smith [18] 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 5).

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 [6], 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 $T_{new} = L + \lambda F(S)$, 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 starts with *tenure* $T = 1$ and it usually maintains a pool Q of the 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 $T = \min\{\max\{1.1T, T+1\}, |V| - 2\}$. Otherwise, the number of iterations without cycles is increased. After a fixed number of iterations without detecting a cycle, we decrement the *tenure* by using the expression $T = \max\{\lfloor 0.9T \rfloor, 1\}$. See Devarenne et al. [2] for a deeper and thorough discussion. The value and effect of these parameters will be studied in the computational experience.

4.3. 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 [9]. 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 that 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 6. It starts by initializing the control variables (steps 1 and 2). Then, the procedure modifies the current solution through moves (steps 3–23). 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*).

Algorithm 6. Pseudo-code of the *diversification* method.

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

Then, if the selected vertex is in S or in $V \setminus S$, the diversification method performs one of the four possible moves (steps 6–17). The diversification process is abandoned if we find an improvement (steps 18–21). Otherwise, it performs iterations until the maximum number of iterations, *maxDiver*, is reached.

As mentioned, our complete tabu search method for the Max Mean Dispersion Problem 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 our method. 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. The Tabu Search method terminates after *maxGlobal* iterations without improving the best solution found.

5. 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 with the 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 [17] 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 1000 (10 instances of Type I and 10 instances of Type II). All these 80 instances are available at http://www.optsicom.es/edp/. Table 1 summarizes their features.

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 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 [21], where the lower the Score, the better the method, and the average CPU time in seconds.

The percentage deviation is computed, for each instance and each method, as the difference between the best-known value, *BKV*, and the value obtained with the method, *Val*, divided by this best known value. In mathematical terms:

$$\text{Dev.} = \frac{BKV - Val}{BKV} * 100$$

As described in Resende et al. [20], for each instance, the *n_score* of a method M is defined as the number of methods that found a better solution than M. In case of ties, all the methods receive the same *n_score*, equal to the number of methods strictly better than all of them. We then report Score as the sum of the *n_score* values across all the instances in the experiment.

Table 2 shows that *Dest* obtains lower average deviation in similar computing times. Additionally, it ranks in the first position in all instances but one (i.e., Score is equal to 1, which means that *Dest* obtains *n_score* = 0 in 19 instances and *n_score* = 1 in only one instance). We therefore select the former as the constructive method for the rest of our experiments.

In the second preliminary experiment, we assess the quality of the local search methods proposed in Section 3. Each local search is executed after the construction of a single solution. Table 3 shows the associated results over the 20 training instances. In this table, we consider the statistics reported in the previous table and additionally, we consider #Best, which indicates the number of times that a method matches the best-known solution.

Results reported in Table 3 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.

In our third preliminary experiment we study the contribution of the short-term memory structure (Section 4.1) in a tabu search method. Initially, we set the parameter tenure to 10 in the 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.

Comparing the results reported in Table 4 with those shown in Table 3, we can see that the tabu search methods systematically produce better outcomes than their memory-less counterpart. Specifically, comparing the best method in Table 3 (LS2) and the ones reported in Table 4 (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 the 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 4.2: 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, $\frac{1}{4}n \leqslant t \leqslant \frac{1}{2}n$; 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 5 these four variant, where 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 for the instances tested.

We complement this experiment by studying the size of the tenure over time. In particular, Fig. 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

Table 1
Instance features.

	Num. of instances	Range of values	Sizes
Type I	40	$[-10, 10]$	150, 500, 750, 1000
Type II	40	$[-10, -5] \cup [5, 10]$	150, 500, 750, 1000

Table 2
Comparison of constructive methods.

	Dev. (%)	Score	Time
Const	11.72	19	0.03
Dest	4.75	1	0.01

Table 3
Comparison of different local search methods.

	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 4
Comparison of different tabu search strategies.

	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 5
Performance of different dynamic tenure approaches.

	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

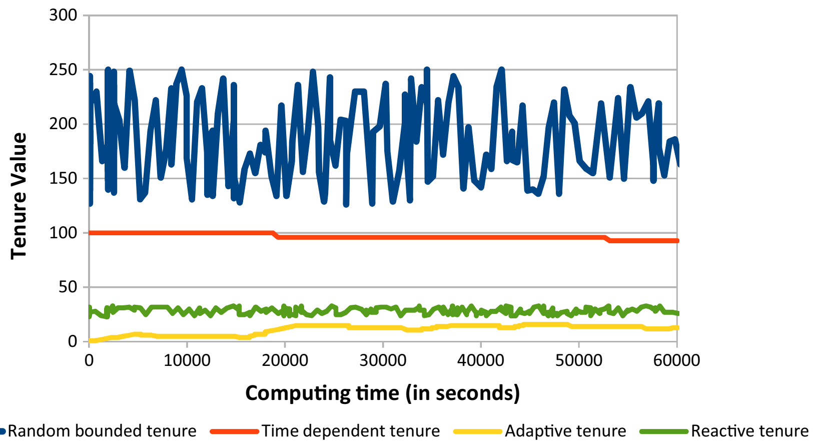

Fig. 2. Tenure evolution over time for one instance.

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.

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 4.3. 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 Tabu Search. These two phases are repeated for *maxGlobal* iterations. Table 6 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 s for all variants.

Table 6 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 5. Therefore, long-term memories must be carefully designed since straightforward implementations could deteriorate the quality of a short-term tabu search.

We use in the final experiment the entire set of 80 instances, divided into 4 subsets of 20 instances each one with size 150, 500, 750, and 1000, respectively. These results are reported in Table 7. In particular, we compare our best tabu search method according to the above experiments (TS_MaxMeanDP_3) with the state-of-the-art algorithm, i.e., the GRASP with Path Relinking (GRASP_PR) introduced in Martí and Sandoya [17]. Additionally, we consider two well-known generic solvers: Gurobi (www.gurobi.com), which is a state-of-the art mathematical programming solver, and Binary Black-Box [12], a context independent solver based on the scatter search methodology, for combinatorial optimization problems with binary variables.

In order to perform a fair comparison, all methods are executed in the same computer for the same total running time (the time elapsed from the data reading until the method completely finishes). We also include in this comparison the average number of evaluations, #Ev., of the objective function (i.e., mean dispersion). This value is independent of the platform in which the method is executed, and complements the running time to evaluate the performance of each method.

The computing time of TS_MaxMeanDP_3 is mainly controlled with the parameter *maxGlobal*. Considering that the size of the instances varies considerably (from 150 to 1000), we set *maxGlobal* to 20, 90, 600, and 1800 s to instances with size 150, 500, 750, and 1000, respectively. We do not set a maximum CPU time for GRASP_PR, since it is executed by using the parameter configuration described in Martí and Sandoya [17]. Gurobi performs an implicit enumeration of the solution space, and therefore it is expected that it can only solve small and medium size problems. As a matter of fact, this solver only was able to target the smallest instances in our test set (those with $n = 150$). Although we run it for a much longer running time than the previous methods (600 s on each instance), the solver did not finish its enumeration process, and its early termination provides a low quality solution (with a 54.92% average deviation from the solution obtained with our tabu search algorithm). Finally, the Black-Box solver is executed in each

Table 6
Testing contribution of different diversification strategies.

	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 7
Comparison with previous methods.

Size	Method	Dev. (%)	#Best	Score	#Ev.	Time
150	GRASP_PR	0.25	3	23	5.34E+06	63.07
	Black_Box	1.69	1	33	2.69E+07	30.01
	TS_MaxMeanDP_3	0.00	17	3	7.46E+04	19.35
500	GRASP_PR	1.45	2	20	1.41E+08	684.11
	Black_Box	2.87	0	38	2.48E+07	90.27
	TS_MaxMeanDP_3	0.02	18	2	4.28E+03	90.43
750	GRASP_PR	0.90	3	18	3.78E+08	3158.29
	Black_Box	3.33	0	39	2.31E+07	601.515
	TS_MaxMeanDP_3	0.02	17	3	9.92E+03	601.165
1000	GRASP_PR	1.03	0	20	9.82E+08	10630.9
	Black_Box	2.75	0	40	2.26E+07	1802.07
	TS_MaxMeanDP_3	0.00	20	0	1.19E+04	1803.99

subset of instances for the maximum computing time described above.

Table 7 reports the results to this experiment. Our method consistently produces better outcomes in all the metrics for the instances tested. In particular, TS_MaxMeanDP_3 consistently outperforms the Black-Box solver in all the metrics considered. It should be noted however that this generic solver could be applied to a wide range of problems (those modeled with binary variables) while our tabu search has been specifically designed to target the MaxMeanDP.

Regarding the comparison with the current state-of-the-art procedure, our method improves the best-known solution in 72 instances (out of 80). Its average deviation across the whole set of instances is 0.01%, which compares favorably with the 1.00% obtained by GRASP_PR. Additionally, our method is much more effective in the search of the local optimum in the set of instances tested. It is on average more than 5 times faster than the current state-of-the-art algorithm (628 vs. 3634 s), and it also exhibits far fewer number of objective function evaluations (about four orders of magnitude).

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-values of $2.61 \times 10^{-4}, 1.9 \times 10^{-5}, 2.67 \times 10^{-5}$, 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 in the performed experiments.

6. Conclusions

Tabu search has nowadays become the method of choice in numerous metaheuristic implementations and practical applications. In this paper, we explore new tabu search strategies in the context of the Max–Mean Diversity problem. 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. It has also been compared with a metaheuristic recently proposed for this problem, and with two previous solvers, Gurobi and a black-box method for combinatorial optimization. The comparisons favor the proposed tabu search implementation.

An additional important goal of this research has been to investigate the influence of some advanced 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 (i.e., an exhaustive exploration of local neighborhoods) and diversification (i.e., a scattered 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; while another long term design, better suited for this problem, is able to efficiently coupled the short term 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) the Government of the Community of Madrid (grant ref. S2013/ICE-2894), and the Generalitat Valenciana (grant refs. ACOMP/2014/A/241 and Prometeo 2013/049).

References

[1] R. Battiti, G. Tecchiolli, The reactive Tabu search, ORSA J. Comput. 6 (2) (1994) 126–140.
[2] I. Devarenne, H. Mabed, A. Caminada, Adaptive tabu tenure computation in local search, in: Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol. 4972, 2008, pp. 1–12.
[3] A. Duarte, R. Martí, Tabu search and GRASP for the maximum diversity problem, Eur. J. Oper. Res. 178 (2007) 71–84.
[4] A. Duarte, R. Martí, F. Glover, F. Gortázar, Hybrid scatter tabu search for unconstrained global optimization, Ann. Oper. Res. 183 (1) (2011) 95–123.
[5] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, J. Glob. Optimiz. 6 (1995) 109–133.
[6] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, J. Comb. Optimiz. 3 (1999) 379–397.
[7] M. Gallego, M. Laguna, R. Martí, A. Duarte, Tabu search with strategic oscillation for the maximally diverse grouping problem, J. Oper. Res. Soc. 64 (2013) 724–734.
[8] M. Gallego, A. Duarte, M. Laguna, R. Martí, Hybrid heuristics for the maximum diversity problem, Comput. Optimiz. Appl. 44 (3) (2009) 411–426.
[9] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
[10] F. Glover, C.C. Kuo, K.S. Dhir, A discrete optimization model for preserving biological diversity, Appl. Math. Model. 19 (1995) 696–701.
[11] F. Glover, C.C. Kuo, K.S. Dhir, Heuristic algorithms for the maximum diversity problem, J. Inform. Optimiz. Sci. 19 (1) (1998) 109–132.
[12] F. Gortázar, A. Duarte, M. Laguna, R. Martí, Black-box scatter search for general classes of binary optimization problems, Comput. Oper. Res. 37 (2010) 1977–1986.
[13] C. Kerchove, P.V. Dooren, The page trust algorithm: how to rank web pages when negative linksare allowed? in: Proceedings SIAM International Conference on Data Mining, 2008, pp. 346–352.
[14] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[15] M. Lozano, A. Duarte, F. Gortázar, R. Martí, A hybrid metaheuristic for the cyclic antibandwidth problem, Knowl. Based Syst. 50 (2013) 103–113.
[16] Z.P. Lü, J.K. Hao, Adaptive tabu search for course timetabling, Eur. J. Oper. Res. 200 (1) (2010) 235–244.
[17] R. Martí, F. Sandoya, GRASP and path relinking for the equitable dispersion problem, Comput. Oper. Res. 40 (12) (2013) 3091–3099.
[18] R. Montemanni, D.H. Smith, A Tabu Search Algorithm with a Dynamic Tabu List for the Frequency Assignment Problem, Technical Report, University of Glamorgan, 2001.
[19] O.A. Prokopyev, N. Kong, D.L. Martinez-Torres, The equitable dispersion problem, Eur. J. Oper. Res. 197 (2009) 59–67.
[20] M. Resende, R. Martí, M. Gallego, A. Duarte, GRASP and path relinking for the max–min diversity problem, Comput. Oper. Res. 37 (2010) 498–508.
[21] C.C. Ribeiro, E. Uchoa, R.F. Werneck, A hybrid GRASP with perturbations for the Steiner problem in graphs, INFORMS J. Comput. 14 (228–246) (2002) 2002.
[22] F.J. Rodríguez, M. Lozano, C. García-Martínez, J.D. González-Barrera, An artificial bee colony algorithm for the maximally diverse grouping problem, Inform. Sci. 230 (2013) 183–196.
[23] T. Wilson, J. Wiebe, P. Hoffmann, Recognizing contextual polarity in phrase-level sentiment analysis, in: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, 2005, pp. 347–354.
[24] B. Yang, W. Cheung, J. Liu, Community mining from signed social networks, IEEE Trans. Knowl. Data Eng. 19 (10) (2007) 1333–1348.