European Journal of Operational Research 200 (2010) 36–44

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A branch and bound algorithm for the maximum diversity problem

Rafael Martí [a,*], Micael Gallego [b], Abraham Duarte [b]

[a] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
[b] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain

A R T I C L E I N F O

Article history:
Received 14 February 2008
Accepted 11 December 2008
Available online 27 December 2008

Keywords:
Maximum diversity problem
Branch and bound
Integer programming

A B S T R A C T

This article begins with a review of previously proposed integer formulations for the maximum diversity problem (MDP). This problem consists of selecting a subset of elements from a larger set in such a way that the sum of the distances between the chosen elements is maximized. We propose a branch and bound algorithm and develop several upper bounds on the objective function values of partial solutions to the MDP. Empirical results with a collection of previously reported instances indicate that the proposed algorithm is able to solve all the medium-sized instances (with 50 elements) as well as some large-sized instances (with 100 elements). We compare our method with the best previous linear integer formulation solved with the well-known software Cplex. The comparison favors the proposed procedure.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

The maximum diversity problem (MDP) consists of selecting a subset of elements from a larger set of elements in such a way that the sum of the distances between the chosen elements is maximized. It has been the subject of study since the early 1990s when Kuo et al. (1993) proposed different linear integer formulations. Over the past 15 years, different heuristic algorithms have been proposed to obtain approximate solutions in short computational times but, as far as we know, no exact method has been considered for this problem.

We can find a large number of MDP applications in different contexts, such as ecological, medical or social sciences, or in animal and plant genetics, in which the search for new varieties via controlled breeding of stocks with diversity traits of interest can be formulated as an MDP (Porter et al., 1975). The maximum diversity problem also appears in the context of ethnicity when studied from a historical perspective, as shown in Swierenga (1977), and more recently in the application of US immigration policies that promote ethnic diversity among immigrants (McConnell, 1988).

Ghosh (1996) proved the NP-completeness of the MDP, proposed a multi-start algorithm and tested it on small instances. In Glover et al. (1998), four different heuristics are introduced for this problem. Since different versions of the MDP include additional constraints, the objective is to design heuristics whose basic moves for transitioning from one solution to another are both simple and flexible, allowing them to be adapted to multiple settings. The authors compare the solutions obtained with their heuristics with the optimal solutions for small instances.

In recent years, researchers have implemented different metaheuristics to obtain high quality solutions to medium and large MDP instances. Katayama and Narihisa (2004) proposed an approximate algorithm based on evolutionary computations. Silva et al. (2004) presented several GRASP algorithms for the MDP and tested them on medium instances. Their extensive computational experiments show that their heuristic procedures outperform previous methods when they are allowed to run for an extremely long time (their methods require on average more than 20 h of CPU time). Andrade et al. (2005) proposed a variant of GRASP by adding a path relinking post-processing, while Santos et al. (2005) hybridize GRASP with data mining methods. Kochenberger and Glover (1999, 2006) reported solving problems with up to 2000 variables with an adaptive memory tabu search method in roughly 16 min. Palubeckis (2007) presented an iterated tabu search and Duarte and Martí (2007) proposed constructive and improvement algorithms for the MDP also based on the tabu search methodology. Finally, Gallego et al. (2009) implemented scatter search for the MDP and showed in the experimentation that it improves over other earlier heuristics.

In Section 2 of this paper we discuss previous formulations for the MDP. Section 3 presents our theoretical contributions, which basically consist of upper bounds on the objective function values of partial solutions, while a description of the exact algorithm is given in Section 4. We put forward our exact solution method, which implements efficient strategies for branching and pruning. The paper closes with a computational study and the associated conclusions.

* Corresponding author. Tel.: +34 96 386 4362.
E-mail addresses: Rafael.Marti@uv.es (R. Martí), Micael.Gallego@urjc.es (M. Gallego), Abraham.Duarte@urjc.es (A. Duarte).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.12.023

2. Previous formulations

Kuo et al. (1993) proposed three linear integer formulations, F1, F2 and F3, to solve the MDP. The authors illustrate the performance of F2 on a real example of small size. However, they did not test these formulations on any set of large problems with different sizes with a mixed integer programming (MIP) solver. The experimentation described in Section 5 compares these three formulations when solving small and medium-sized MDP instances with Cplex 8.0.

The MDP consists of selecting a subset of m elements from a set of n elements in such a way that the sum of the distances between the chosen elements is maximized. In most applications, it is assumed that each element can be represented by a set of K attributes. Let s_{ik} be the state or value of the kth attribute of element i, where $i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, K$. Then, the distance d_{ij} between elements i and j may be simply defined as the Euclidean distance as shown in the mathematical expression below, or as any other distance function

$$d_{ij} = \sqrt{\sum_{k=1}^{K} (s_{ik} - s_{jk})^2}.$$

The maximum diversity problem can then be formulated as the following quadratic zero-one integer program, where variable x_i takes the value 1 if element i is selected and 0 otherwise, $i = 1, 2, \ldots, n$.

$$\begin{aligned}
\text{(F1)}\quad \text{Maximize}\quad & z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j \\
\text{Subjectto}\quad & \sum_{i=1}^{n} x_i = m, \\
& x_i = \{0, 1\}, \quad 1 \leqslant i \leqslant n.
\end{aligned}$$

Glover and Woolsey (1974) show how to convert a 0–1 polynomial programming problem into a 0–1 linear programming problem by replacing the product of some variables with a new variable satisfying some additional constraints. Kuo et al. (1993) apply this transformation and replace the product $x_i x_j$ in F1 with a new variable y_{ij}, obtaining the mixed integer program F2 below, in which the additional constraints have been included

$$\begin{aligned}
\text{(F2)}\quad \text{Maximize}\quad & z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} y_{ij} \\
\text{Subjectto}\quad & \sum_{i=1}^{n} x_i = m, \\
& x_i + x_j - y_{ij} \leqslant 1, \quad 1 \leqslant i < j \leqslant n, \\
& -x_i + y_{ij} \leqslant 0, \quad 1 \leqslant i < j \leqslant n, \\
& -x_j + y_{ij} \leqslant 0, \quad 1 \leqslant i < j \leqslant n, \\
& y_{ij} \geqslant 0, \quad 1 \leqslant i < j \leqslant n, \\
& x_i = \{0, 1\}, \quad 1 \leqslant i \leqslant n.
\end{aligned}$$

Glover (1975) introduces some inequalities to handle quadratic integer programs with both real and binary variables. Kuo et al. (1993) apply this transformation to F1 by decomposing the objective function into the sum of w-values

$$z = \sum_{i=1}^{n-1} \left(x_i \sum_{j=i+1}^{n} d_{ij} x_j \right) = \sum_{i=1}^{n-1} w_i.$$

Define the following:

$$\overline{D_i} = \sum_{j=i+1}^{n} \max(0, d_{ij}) \quad \text{and} \quad \underline{D_i} = \sum_{j=i+1}^{n} \min(0, d_{ij}).$$

F1 is reformulated as F3

$$\begin{aligned}
\text{(F3)}\quad \text{Maximize}\quad & z = \sum_{i=1}^{n-1} w_i \\
\text{Subjectto}\quad & \sum_{i=1}^{n} x_i = m, \\
& -\overline{D_i} x_i + w_i \leqslant 0, \quad 1 \leqslant i \leqslant n-1, \qquad (1) \\
& -\sum_{j=i+1}^{n} d_{ij} x_j + \underline{D_i}(1 - x_i) + w_i \leqslant 0, \quad 1 \leqslant i \leqslant n-1, \qquad (2) \\
& x_i = \{0, 1\}, \quad 1 \leqslant i \leqslant n. \qquad (3)
\end{aligned}$$

We can see how the new constraints make w_i take the appropriate value in order to obtain the same z value as in F1. For example, when $x_i = 0$, (1) implies $w_i \leqslant 0$. In view of (2), we have:

$$-\sum_{j=i+1}^{n} d_{ij} x_j + \underline{D_i} + w_i \leqslant 0,$$

which implies that $w_i \leqslant \sum_{j=i+1}^{n} d_{ij} x_j - \underline{D_i}$. By definition,

$$\underline{D_i} \leqslant \sum_{j=i+1}^{n} d_{ij} x_j.$$

Therefore, (2) is less restrictive than (1) in F3 and the maximum value that w_i can take to maximize the objective function z is 0. In short, if $x_i = 0$ then $w_i = 0$ and each of their contributions to the objective function is 0. Similarly, we can see that if $x_i = 1$ then

$$w_i = \sum_{j=i+1}^{n} d_{ij} x_j.$$

So F3 and F1 provide the same objective value and both problems are equivalent. Therefore, the maximum diversity problem can be formulated as F1, F2 or F3. In the computational experiments reported in Section 5 we will see their effectiveness when solving a set of instances of the MDP.

3. Upper bounds for partial solutions

Consider the underlying graph in the MDP definition where each element is represented as a vertex, and the distance between each pair of elements as the weight or cost associated with the corresponding edge. The MDP then consists of selecting a subset of vertices in such a way that the sum of the weights between them is maximized. In this section we prove some upper bounds on the objective function values of a set of solutions to the MDP that share common vertices, which will be called *partial solutions*. We start by defining a partial solution and splitting the objective function value associated with it into three parts. It will enable us to compute the three upper bounds UB_2, UB_3 and UB_{23} introduced in Propositions 1–3 of this section, respectively.

Let V be the set of vertices of a graph G and let $m < n$ be the number of vertices that we have to select in an MDP solution. We define a partial solution Sel as a set of k vertices in V with $k < m$. The set $Sel \subset V$ can be viewed as an incomplete solution. We can obtain a complete solution to the MDP by adding $m - k$ elements from $V \backslash Sel$ to it. Let S_{Sel} be the set of all solutions obtained by adding elements to Sel (i.e. the set of solutions in which all the elements in Sel are selected).

Consider a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ and a complete solution $x = \{s_1, s_2, \ldots, s_k, u_1, u_2, \ldots, u_{m-k}\}$ in S_{Sel}. We denote the set of vertices in x not present in Sel as $Unsel(x) = \{u_1, u_2, \ldots, u_{m-k}\}$. The objective function value z associated with x can be broken down into $z = z_1 + z_2 + z_3$, where z_1 is the sum of the distances (edge weights) between the pairs of selected vertices (vertices in *Sel*), z_2 is the sum of the edge weights with one extreme in *Sel* and the other in *Unsel*(x), and z_3 is the sum of edge weights with both extremes in *Unsel*(x). More specifically,

$$z_1 = \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} d(s_i, s_j), \qquad z_2 = \sum_{i=1}^{k}\sum_{j=1}^{m-k} d(s_i, u_j),$$
$$z_3 = \sum_{i=1}^{m-k-1}\sum_{j=i+1}^{m-k} d(u_i, u_j).$$

Note that in this section $d(a, b)$ represents the distance between elements a and b, denoted as d_{ab} in the previous section. We use the example given in Fig. 1 and the associated Euclidean distance matrix in Table 1 to illustrate these concepts. The dimensions of the example problem are $n = 6$ and $m = 4$.

Consider the partial solution $Sel = \{1, 3\}$ in the example given in Table 1. We have $S_{Sel} = \{\{1, 3, 2, 4\}, \{1, 3, 4, 5\}, \{1, 3, 5, 6\}, \{1, 3, 2, 5\}, \{1, 3, 4, 6\}, \{1, 3, 2, 6\}\}$. Consider for example the complete solution $x = \{1, 3, 5, 6\}$ in S_{Sel}. Then we have $Unsel(x) = \{5, 6\}$. We can compute the objective function value of x as $z = z_1 + z_2 + z_3 = 3.16 + 20.27 + 3.00 = 26.43$ where

$$\begin{aligned} z_1 &= d(1,3) = 3.16,\\ z_2 &= d(1,5) + d(3,5) + d(1,6) + d(3,6)\\ &= 5.66 + 5.10 + 4.12 + 5.39 = 20.27,\\ z_3 &= d(5,6) = 3.00. \end{aligned}$$

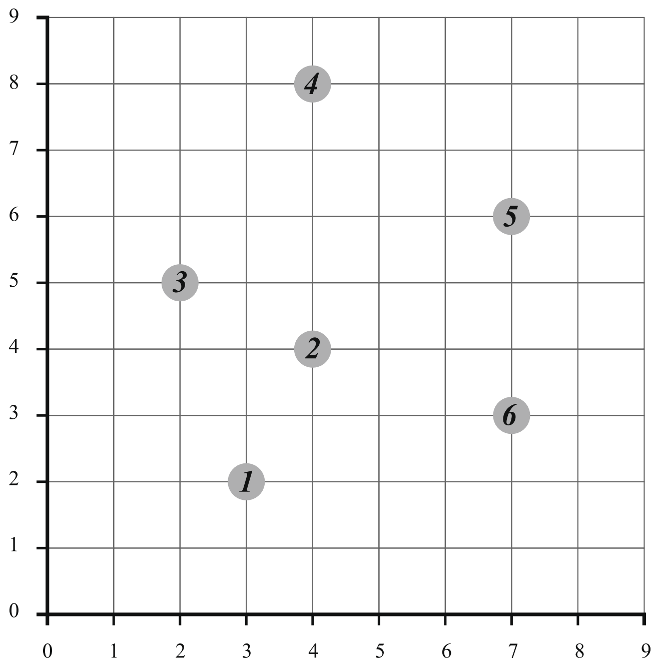

Fig. 1. MDP example.

Table 1
Distance matrix.

	1	2	3	4	5	6
1	–	2.24	3.16	6.08	5.66	4.12
2	2.24	–	2.24	4.00	3.61	3.16
3	3.16	2.24	–	3.61	5.10	5.39
4	6.08	4.00	3.61	–	3.61	5.83
5	5.66	3.61	5.10	3.61	–	3.00
6	4.12	3.16	5.39	5.83	3.00	–

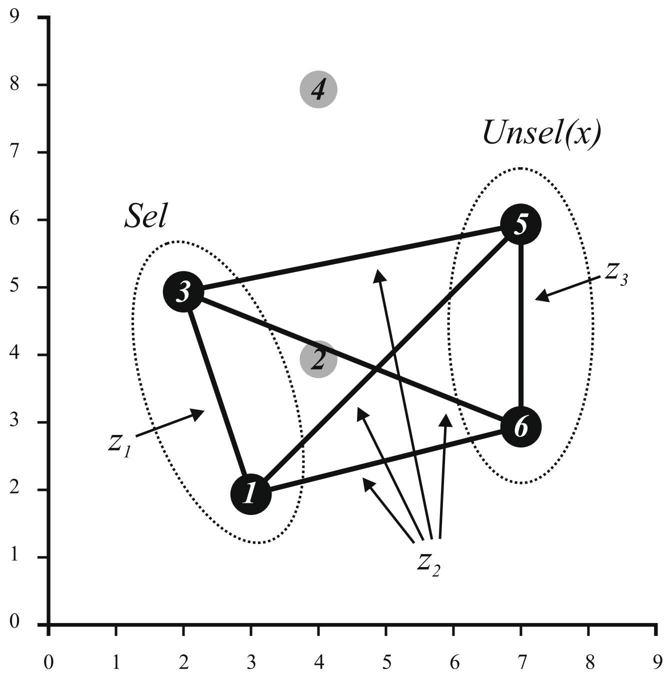

Fig. 2. Decomposition of the solution value.

Fig. 2 illustrates the decomposition of the value z of the solution $x = \{1, 3, 5, 6\}$. It is clear that for any solution in S_{Sel}, z_1 is an invariant and in this example it takes the value of 3.16. In this section we want to obtain an upper bound of the objective function value of any solution in S_{Sel}. Propositions 1 and 2 provide upper bounds for the objective solution values of z_2 and z_3 respectively for any solution in S_{Sel}. Proposition 3 provides an upper bound on the value of $z_2 + z_3$.

Given a set of vertices, $Sel = \{s_1, s_2, \ldots, s_k\}$, and a vertex $v \in V \backslash Sel$, we define $z_{Sel}(v)$ as

$$z_{Sel}(v) = \sum_{i=1}^{k} d(s_i, v),$$

we can interpret $z_{Sel}(v)$ as the potential contribution of v with respect to the selected vertices if we add it to the partial solution *Sel*.

Proposition 1. *Given a partial solution to the MDP* $Sel = \{s_1, s_2, \ldots, s_k\}$ *with* $k < m$, *let* $z^1_{Sel}, z^2_{Sel}, \ldots, z^{n-k}_{Sel}$ *be values of* $z_{Sel}(v)$ *in descending order,* $v \in V \backslash Sel$*; then, for any complete solution* x *in* S_{Sel}

$$z_2 \leqslant UB_2 = \sum_{j=1}^{m-k} z^j_{Sel}.$$

Proof. Given the solution $x = \{s_1, s_2, \ldots, s_k, u_1, u_2, \ldots, u_{m-k}\}$ in S_{Sel}, it is clear that

$$z_2 = \sum_{i=1}^{k}\sum_{j=1}^{m-k} d(s_i, u_j) = \sum_{j=1}^{m-k} z_{Sel}(u_j) \leqslant \sum_{j=1}^{m-k} z^j_{Sel}. \qquad \square$$

Given a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ with $k < m$, let $d^1_{Unsel}(v), d^2_{Unsel}(v), \ldots, d^{n-k-1}_{Unsel}(v)$ be the distances between v in $V \backslash Sel$ and the other vertices in $V \backslash Sel$ in descending order. We define $z_{Unsel}(v)$ as

$$z_{Unsel}(v) = \frac{1}{2}\sum_{i=1}^{m-k-1} d^i_{Unsel}(v),$$

we can interpret $z_{Unsel}(v)$ as an upper bound on the potential contribution of v with respect to the unselected vertices if we add it to the partial solution *Sel* under construction.

In the example in Fig. 1 and Table 1 with $Sel = \{1, 3\}$, if we consider vertex 2 then we have $d(2, 4) = 4.00$, $d(2, 5) = 3.61$ and $d(2, 6) = 3.16$. It follows that

$$d^1_{Unsel}(2) = 4.00, \quad d^2_{Unsel}(2) = 3.61,$$
$$d^3_{Unsel}(2) = 3.16 \quad \text{and} \quad z_{Unsel}(2) = \frac{1}{2}(4.00) = 2.00.$$

Proposition 2. *Given a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ with $k < m$, let $z^1_{Unsel}, z^2_{Unsel}, \ldots, z^{n-k}_{Unsel}$ be the values of $z_{Unsel}(v)$ in descending order, $v \in V \backslash Sel$; then for any complete solution x in S_{Sel}*

$$z_3 \leqslant UB_3 = \sum_{j=1}^{m-k} z^j_{Unsel}.$$

Proof. Given the solution $x = \{s_1, s_2, \ldots, s_k, u_1, u_2, \ldots, u_{m-k}\}$ in S_{Sel}, it is clear that

$$z_3 = \sum_{i=1}^{m-k-1} \sum_{j=i+1}^{m-k} d(u_i, u_j) = \frac{1}{2} \sum_{i=1}^{m-k} \sum_{j=1}^{m-k} d(u_i, u_j)$$
$$\leqslant \sum_{i=1}^{m-k} z_{Unsel}(u_i) \leqslant \sum_{i=1}^{m-k} z^i_{Unsel}. \qquad \square$$

Consider again the partial solution $Sel = \{1,3\}$ in the example given in Table 1. Applying Propositions 1 and 2 we can bound the value z of any solution in S_{Sel} as

$$z = z_1 + z_2 + z_3 \leqslant z_1 + UB_2 + UB_3 = 3.16 + 20.45 + 5.83 = 29.44.$$

It is easy to see that 29.44 is larger than the objective function value associated with any complete solution involving $Sel = \{1, 3\}$. Specifically, we denote the objective function value of a complete solution x as $z(x)$. It is then seen that

$$z(\{1,3,2,4\}) = 21.33,$$
$$z(\{1,3,2,5\}) = 22.01,$$
$$z(\{1,3,2,6\}) = 20.31,$$
$$z(\{1,3,4,6\}) = 28.19,$$
$$z(\{1,3,4,5\}) = 27.22,$$
$$z(\{1,3,5,6\}) = 26.43.$$

Note that UB_2 and UB_3 are computed independently. We can improve the final bound if we simply merge both values. Given a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ with $k < m$ for any v in $V \backslash Sel$, we define $z(v)$ as

$$z(v) = z_{Sel}(v) + z_{Unsel}(v).$$

Proposition 3. *Given a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ with $k < m$, let $z^1, z^2, \ldots, z^{n-k}$ be the values of $z(v)$ in descending order, $v \in V \backslash Sel$; then, for any solution x in S_{Sel}*

$$z_2 + z_3 \leqslant UB_{23} = \sum_{j=1}^{m-k} z^j.$$

Proof. We only need to put the proofs of Propositions 1 and 2 together to obtain the result sought

$$z_2 + z_3 = \sum_{j=1}^{k} \sum_{i=1}^{m-k} d(s_j, u_i) + \sum_{i=1}^{m-k-1} \sum_{j=i+1}^{m-k} d(u_i, u_j)$$
$$= \sum_{j=1}^{k} \sum_{i=1}^{m-k} d(s_j, u_i) + \frac{1}{2} \sum_{i=1}^{m-k} \sum_{j=1}^{m-k} d(u_i, u_j),$$
$$z_2 + z_3 \leqslant \sum_{i=1}^{m-k} z_{Sel}(u_i) + \sum_{i=1}^{m-k} z_{Unsel}(u_i) = \sum_{i=1}^{m-k} z(u_i) \leqslant \sum_{i=1}^{m-k} z^i = UB_{23}. \qquad \square$$

In the example above we have

$$z(2) = z_{Sel}(2) + z_{Unsel}(2) = 4.48 + \max\left(\frac{d(2,4)}{2}, \frac{d(2,5)}{2}, \frac{d(2,6)}{2}\right)$$
$$= 4.48 + \max\left(\frac{4.00}{2}, \frac{3.61}{2}, \frac{3.16}{2}\right) = 4.48 + \frac{4.00}{2} = 6.48.$$

Similarly,

$$z(4) = 9.69 + \max\left(\frac{4.00}{2}, \frac{3.61}{2}, \frac{5.83}{2}\right) = 12.61,$$
$$z(5) = 10.76 + \max\left(\frac{3.61}{2}, \frac{3.61}{2}, \frac{3.00}{2}\right) = 12.57,$$
$$z(6) = 9.51 + \max\left(\frac{3.16}{2}, \frac{5.83}{2}, \frac{3.00}{2}\right) = 12.43.$$

Therefore the improved bound given in Proposition 3 for this example is $z_1 + UB_{23} = 3.16 + 12.61 + 12.57 = 28.34$, which is better (lower) than the bound of 29.44 obtained with Propositions 1 and 2. Note that it is easy to see that $UB_{23} \leqslant UB_2 + UB_3$ because UB_2 is computed with respect to the maximum $z_{Sel}(v)$ values, UB_3 is computed with respect to the maximum $z_{Unsel}(v)$ values and UB_{23} is computed with respect to the vertices for which $z_{Sel}(v) + z_{Unsel}(v)$ is maximized. Note that the vertices involved in computing UB_2 and those involved in computing UB_3 may be different, while we use the same vertices in $z_{Sel}(v)$ and $z_{Unsel}(v)$ when computing UB_{23} and therefore its maximum is lower than or equal to $UB_2 + UB_3$.

As mentioned, we compute UB_{23} as $z(4) + z(5)$ because vertices 4 and 5 have the largest and the second largest z-values, respectively. Consider now the solution x in which 4 and 5 are selected ($x = \{1, 3, 4, 5\}$). It has an objective function value of $z = 27.22$ strictly lower than the bound $z_1 + UB_{23} = 28.34$, which means that no solution in S_{Sel} (where $Sel = \{1, 3\}$) has an objective function value of 28.34. In this situation (when the solution corresponding to the vertices with largest z-values has a value strictly lower than the upper bound) we can apply the following procedure to tighten this upper bound. We can compute UB_{23} excluding vertex 4, $UB_{23}(4)$, and then compute UB_{23} excluding vertex 5, $UB_{23}(5)$. The maximum between both values plus z_1 and the objective function value associated with the complete solution x, z, is an improved upper bound, IUB. In our example

$$IUB = \max\{z, z_1 + UB_{23}(4), z_1 + UB_{23}(5)\}$$
$$= \max\{27.22, 28.16, 28.2\} = 28.2.$$

Given a partial solution $Sel = \{s_1, s_2, \ldots, s_k\}$ we first compute the upper bound $z_1 + UB_{23}$. We then consider the complete solution x in which the vertices with maximum z-values, $v^*_1, v^*_2, \ldots, v^*_{m-k}$, involved in the computation of UB_{23}, are selected (v^*_i is the vertex associated with z^i in computing UB_{23}, $i = 1, 2, \ldots, m-k$). Note that this is a simple heuristic to generate a good solution (in which "promising" vertices are selected). We test whether this solution improves or not the incumbent solution (global lower bound). Finally, if the value of this solution is strictly lower than the upper bound ($z < z_1 + UB_{23}$), we calculate IUB by simply computing the maximum between $z, z_1 + UB_{23}(v^*_1), z_1 + UB_{23}(v^*_2), \ldots,$ and $z_1 + UB_{23}(v^*_{m-k})$. Note that although this calculation implies an extra computational effort it is only performed for a small fraction of all the vertices.

4. The branch and bound algorithm

Given a graph $G = (V, E)$ with a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and letting $m < n$ be the number of vertices that we have to select in a complete solution, the search tree to be described below provides a generation and partition of the set of solutions for the maximum diversity problem in which each node represents a partial solution (except the *leaves* or final nodes that represent complete solutions).

Branch and bound is a general algorithm for finding optimal solutions to optimization problems. It consists of a systematic enumeration of all solutions, where a large number of them are discarded by using upper and lower bounds of their objective function value. In our branch and bound algorithm for the MDP we start by running a heuristic algorithm to obtain an initial solution, whose objective function value gives a lower bound for the MDP (since it is a maximization problem). In the first version of our method, the search tree is explored in a depth first manner. The upper bound $z_1 + UB_{23}$ is computed at each node. If it is lower than the lower bound we fathom the node (because no better solution can be found in this node than the incumbent one); otherwise we branch the node and explore its first *child* node. When the exploration reaches a *leaf* node it computes the objective function value of the associated complete solution, and updates the lower bound if necessary. Then it performs a backward step, checking again its *parent* node (backtracking) with the new lower bound and continuing the exploration. The branch and bound algorithm stops when all the nodes have been examined (some of them have been branched and others fathomed), and returns as the output the optimum of the problem. An early termination, due to time limitations, provides us with a lower bound and an upper bound of the optimum (this latter bound is obtained as the maximum of the upper bounds in the unexplored nodes).

4.1. Search tree

As a first step towards solving the MDP, we build the complete enumeration tree as follows. The initial node branches into n nodes (labeled from 1 to n, where node i represents the partial solution $Sel = \{i\}$). Each of these n nodes in the first level branches into $n - 1$ nodes (which will be referred to as nodes in level 2). For instance, node 2 in the first level ($Sel = \{2\}$) has $n - 1$ successors in level 2 (labeled as $1, 3, 4, \dots, n$). So, node 3 in the second level, the successor of node 2 in the first level, represents the partial solution $Sel = \{2, 3\}$. Therefore, at each level in the search tree, the algorithm extends the current partial solution by adding one vertex. Fig. 3 represents this basic search tree for an example with $n = 5$ and $m = 3$.

As shown in Fig. 3, the basic search tree described above contains repetitions of the same solutions. For example, the nodes marked as A, B, C and D all represent the same solution $\{1, 2, 3\}$. Therefore, this tree does not represent a partition of the set of complete solutions to the MDP. We then consider a search tree with no repetitions. Specifically, the initial node branches into $n - m + 1$ nodes (labeled from 1 to $n - m + 1$, where node i represents the partial solution $Sel = \{i\}$). Each of these $n - m + 1$ nodes in the first level branches into a number of nodes in level 2 that depends on the node label. In general, node$i$ in level k branches into $n - (m - k) - i + 1$ nodes, beginning with node $i + 1$ and ending with node $n - (m - k) + 1$. In the example above with $n = 5$ and $m = 3$, Fig. 4 shows this search tree with no repetitions. Node 1 in level 1 branches into nodes 2, 3 and 4; node 2 in level 1 branches into nodes 3 and 4, and node 3 in level 1 branches into node 4. Node 2 in level 2 branches into nodes 3, 4 and 5; nodes 3 branch into nodes 4 and 5, and nodes 4 branch into node 5. The nodes in level 3, *leaf* nodes, represent complete solutions that are shown in the right hand column. Comparing the search tree in Fig. 4 with that in Fig. 3, we can see that the size has been substantially reduced by avoiding the repetitions. We implement the construction of the search tree by means of the algorithm *GenerateNode*, with specialized data structures that make our final procedure efficient.

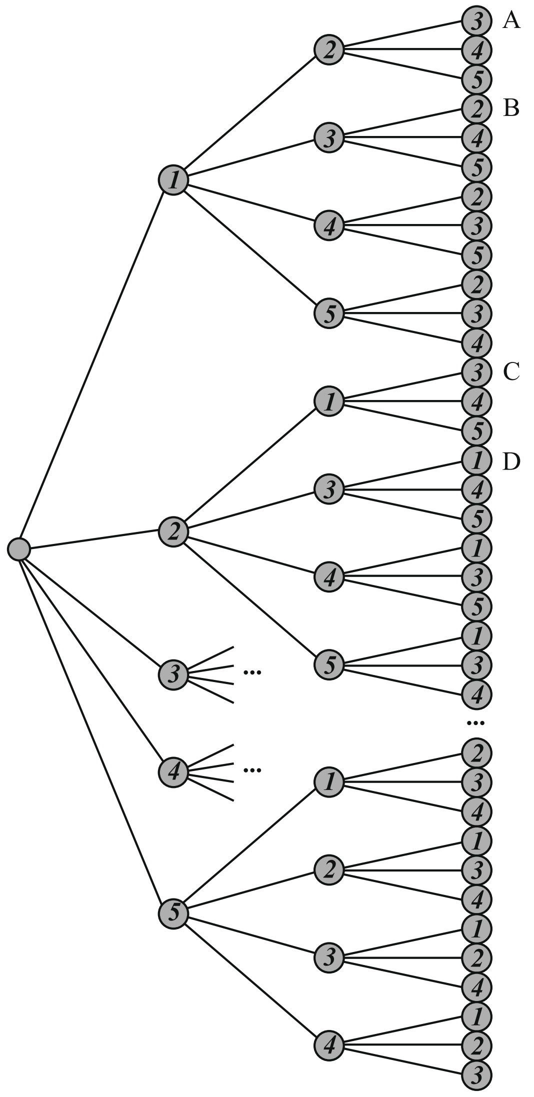

Fig. 3. Basic search tree.

The argument of *GenerateNode* is the level k in the search tree and it is associated with the position in an integer array to insert the new element in the solution under construction. In this way the solution is constructed step by step in each call to this function.

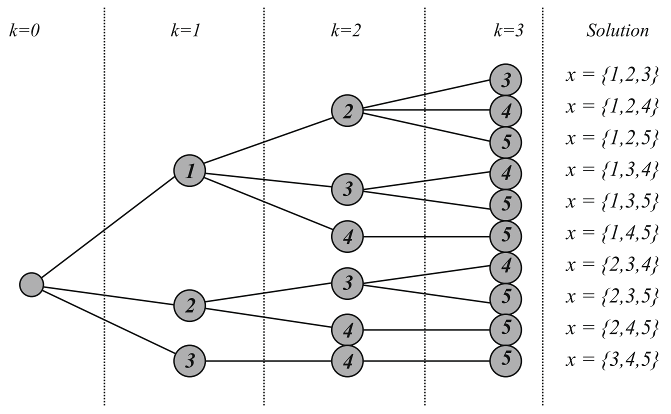

Fig. 4. Improved search tree.

4.2. Improved exploration

In this subsection we describe a second test to fathom nodes in the search tree based on a new result presented in Proposition 4. Given a complete solution $x = \{s_1, s_2, \ldots, s_m\}$, we can express its objective function value as

$$z = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} d(s_i, s_j) = \sum_{i=1}^{m} d(s_i),$$

where $d(s_i) = \frac{1}{2}\sum_{j=1}^{m} d(s_i, s_j)$ represents the contribution of s_i to the value z. From this expression we can introduce the maximum and the minimum potential contributions of a vertex v to any solution, $d_{\max}(v)$ and $d_{\min}(v)$. Let $d(v, v_{\sigma(1)}), d(v, v_{\sigma(2)}), \ldots, d(v, v_{\sigma(n-1)})$ be the weights along the edges connecting v and the other vertices in the graph in descending order. It follows that

$$d_{\min}(v) = \frac{1}{2}\sum_{j=1}^{m-1} d(v, v_{\sigma(n-j)}), \quad d_{\max}(v) = \frac{1}{2}\sum_{j=1}^{m-1} d(v, v_{\sigma(j)}).$$

It is clear from the above definition that any solution containing vertex v will satisfy the following inequalities:

$$d_{\min}(v) \leqslant d(v) \leqslant d_{\max}(v).$$

Proposition 4. *Given an optimal solution x and two vertices u, $v \in V$, if $d_{max}(u) < d_{min}(v)$ and v is not selected in x, then u cannot be selected in x.*

Proof. Without loss of generality, let us assume that u is in the optimal solution $x = \{u, s_2, s_3, \ldots, s_m\}$ and we will see that it leads us to a contradiction. The objective function value z associated with x can be broken down into two parts as follows:

$$z = \sum_{j=2}^{m} d(u, s_j) + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} d(s_i, s_j).$$

From the definition of $d_{\max}$ we have

$$z = \sum_{j=2}^{m} d(u, s_j) + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} d(s_i, s_j) \leqslant 2d_{\max}(u) + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} d(s_i, s_j).$$

Similarly, if we consider the solution $y = \{v, s_2, s_3, \ldots, s_m\}$ with an objective value of z'. Then we have

$$z' = \sum_{j=2}^{m} d(v, s_j) + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} d(s_i, s_j) \geqslant 2d_{\min}(v) + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} d(s_i, s_j).$$

Applying $d_{\max}(u) < d_{\min}(v)$ in the two previous inequalities, we obtain $z' > z$. This contradicts the fact that x is an optimal solution; therefore, u cannot be selected in an optimal solution in which v is not selected. □

Fig. 5 illustrates the application of Proposition 4 to fathom some nodes in the search tree in an example with $n = 6$ and $m = 3$. Let us assume here that $d_{\max}$ (3) < $d_{\min}$ (1). Then, according to Proposition 4, if node 1 is not selected in an optimal solution, vertex 3 cannot be selected. Therefore, we can fathom all the solutions (partial and complete) in which vertex 3 is selected and vertex 1 is not, because they cannot be optimal solutions. Fig. 5 shows the search tree of this example in which 10 nodes (depicted with dashed lines) are fathomed. Note that one of the fathomed nodes is in the first level and three of them are in the second one.

As we have mentioned, at each node of the search tree we compute $z_1 + UB_{23}$ and test whether we can fathom the node or not. To calculate UB_{23}, we first need to obtain $z(v) = z_{Sel}(v) + z_{Unsel}(v)$ for

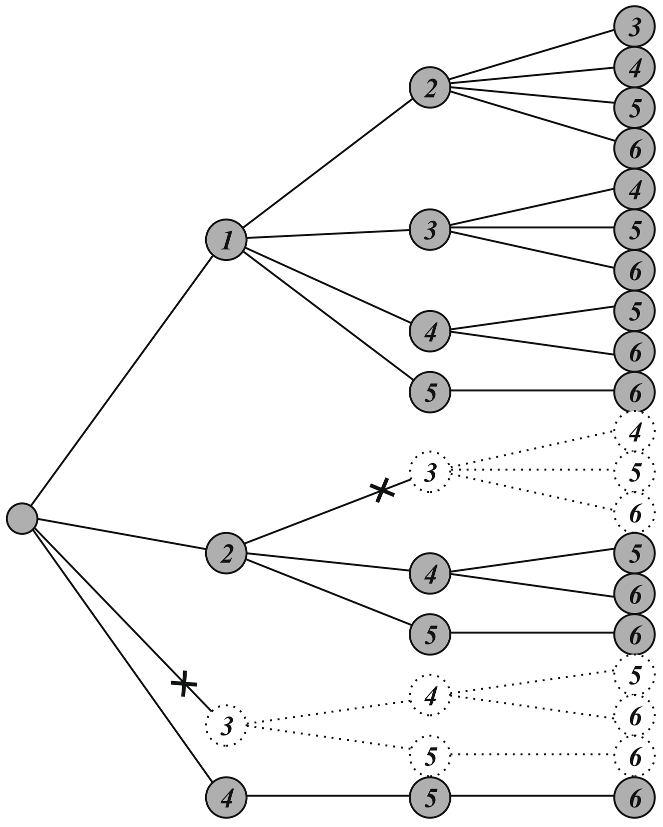

Fig. 5. Fathomed nodes in search tree.

any v in $V \backslash Sel$, where $z_{Unsel}(v)$ involves several time-consuming calculations. However, we have found that the way in which the search tree is generated to avoid duplications allows us to compute $z_{Unsel}(v)$ offline. Specifically, $z_{Unsel}(v)$ only depends on the value of v, k (number of selected vertices) and, in this search tree, on the largest label among the selected vertices (*max_label*). For example, in the search tree shown in Fig. 5, we can see that the two nodes labeled with 3 in level 2 have the same values for $z_{Unsel}(4)$, $z_{Unsel}(5)$ and $z_{Unsel}(6)$ because $k = 2$ and *max_label* = 3. The same happens with the nodes labeled as 4 and 5 in level 2 of Fig. 5. Therefore, we compute the values of $z_{Unsel}(v)$ for each possible combination of k and *max_label* at the beginning of the implementation of the method. In the next section we empirically show the effectiveness of this offline computation.

Fig. 6 shows a pseudo-code of the complete branch and bound algorithm. To make it short we introduce the set *ActiveNodes* in which the nodes in the search tree are stored, selected and removed according to the function *GenerateNode* described above.

5. Computational experiments

This section describes the computational experiments that we performed to test the efficiency of our branch and bound procedure as well as compare it with the linear integer formulations proposed previously. We have implemented the branch and bound algorithm in Java SE 6 and solved the integer formulations with Cplex 8.0. All the experiments were conducted on a Pentium 4 computer at 3 GHz with 3 GB of RAM.

We have employed three sets of instances in our experimentation. The first one was introduced in Glover et al. (1998), the second one, *Glover*2, is an extension of the first one to target large MDP graphs, and the third set is from Silva et al. (2004):

```
1. Compute an initial lower bound $LB$ with a heuristic algorithm
2. Make $ActiveNodes$={ Initial node and its child nodes in the search tree}
While ($ActiveNodes \neq \emptyset$)
    3. Take $Node$ from $ActiveNodes$
    If ($Node$ is a leaf node)
        4. Compute the value z' of its associated solution
        If ($z' > LB$)
            5. $LB = z'$
        6. Remove $Node$ from $ActiveNodes$
    Else
        7. Let $Sel = \{s_1, s_2, \ldots, s_k\}$ be the partial solution associated with $Node$
        8. Compute $d_{\max}(s_j)$ for $j = 1, 2, \ldots, k$ and $z(v)$ as well as $d_{\min}(v)$ for $v \in V \backslash Sel$
        If($d_{\max}(s_j) < d_{\min}(v)$ for any $j = 1, 2, \ldots, k$ and $v \in V \setminus Sel$ )
            9. Remove $Node$ from $ActiveNodes$
        Else
            10. Compute the upper bound $z_1 + UB_{23}$ of $Node$
            11. Let $v_1^*, v_2^*, \ldots, v_{m-k}^*$ be the vertices in $V \setminus Sel$ with maximum $z$-values
            12. Compute the value $z'$ of solution $x = \{s_1, s_2, \ldots, s_k, v_1^*, v_2^*, \ldots, v_{m-k}^*\}$
            If ($z' > LB$)
                13. $LB = z'$
            If ($z_1 + UB_{23} < LB$)
                14. Remove $Node$ from $ActiveNodes$
            Else If ($z' < z_1 + UB_{23}$)
                15. $IUB = \max(z, z_1 + UB_{23}(v_1^*), z_1 + UB_{23}(v_2^*), \ldots, z_1 + UB_{23}(v_{m-k}^*))$
                If ($IUB < LB$)
                    16. Remove $Node$ from $ActiveNodes$
            Else
                17. Add the child nodes of $Node$ to $ActiveNodes$
```

Fig. 6. Branch and bound algorithm pseudo-code.

Glover: *This data set consists of 75 matrices for which the values were calculated as the Euclidean distances from randomly generated points with coordinates in the 0–100 range. The number of coordinates for each point is also randomly generated between 2 and 21. Glover et al. (1998) developed this data generator and constructed instances with n = 10, 15 and 30. The value of m ranges from 0.15n to 0.75n.*

Glover2: *This data set consists of 50 matrices constructed with the same graph generator employed in the Glover set. We generated 10 instances with n = 25, 50, 100, 125 and 150. For each value of n we consider m = 0.1n and 0.3n (generating five instances for each combination of n and m).*

Silva: *This data set consists of 50 matrices with random numbers between 0 and 9 generated from an integer uniform distribution. These instances were introduced by Silva et al. (2004). We use their generator to construct instances of the same size as those in the Glover2 set (in order to compare the performance of the methods in both sets).*

In our first experiment we compare the three linear integer formulations, F1, F2 and F3, introduced in Section 2. We limit the execution of the Cplex solver to 1 h (3600 s) of computer time. For each combination of the n and m values in the *Glover* set of instances, Table 2 shows the average gap, Gap, the CPU time in seconds, CPU, and the number of optima that each method is able to match, #Opt. The average gap is computed as the upper bound minus the best solution found (both returned by Cplex when the time limit is reached), divided by the upper bound and multiplied by 100.

Table 2 shows that formulation F3 produces better results than F1 and F2 with respect to CPU time. Specifically, the application of F1 and F2 leads to average gaps of 0.7% and 0.8% and the computer solution times are 591.6 and 711.7 s on average respectively, while the application of F3 leads to an average gap of 0.0% and the computer solution time is 96.9 s on average. In addition, F1, F2 and F3 obtain 65, 66 and 75 optimal solutions respectively (out of 75 instances).

In the following experiments we test our branch and bound method. In line with testing the effectiveness of the branch and

Table 2
Results based on three linear integer formulations.

n	m	F1			F2			F3		
		Gap	CPU	#Opt	Gap	CPU	#Opt	Gap	CPU	#Opt
10	2	0.0	0.0	5	0.0	0.1	5	0.0	0.0	5
10	3	0.0	0.0	5	0.0	0.1	5	0.0	0.0	5
10	4	0.0	0.0	5	0.0	0.1	5	0.0	0.0	5
10	6	0.0	0.0	5	0.0	0.1	5	0.0	0.0	5
10	8	0.0	0.0	5	0.0	0.1	5	0.0	0.0	5
15	3	0.0	0.0	5	0.0	0.6	5	0.0	0.1	5
15	4	0.0	0.1	5	0.0	0.8	5	0.0	0.1	5
15	6	0.0	0.4	5	0.0	1.3	5	0.0	0.2	5
15	9	0.0	0.4	5	0.0	1.1	5	0.0	0.2	5
15	12	0.0	0.0	5	0.0	0.6	5	0.0	0.0	5
30	6	0.0	59.0	5	0.0	514.7	5	0.0	35.7	5
30	9	0.0	1597.0	5	4.7	3564.7	1	0.0	292.9	5
30	12	7.6	3600.0	0	6.8	3501.9	1	0.0	929.3	5
30	18	3.5	3600.0	0	1.0	3055.9	4	0.0	193.2	5
30	24	0.0	16.8	5	0.0	33.7	5	0.0	1.9	5
Average		0.7	591.6	4.3	0.8	711.7	4.4	0.0	96.9	5

Table 3
Results with and without offline calculation.

	n	m	Without offline calculation		With offline calculation	
			CPU	#Opt	CPU	#Opt
*Glover*2	50	5	0.9	5	0.07	5
	50	15	53.98	5	3.58	5
Silva	50	5	0.17	5	0.02	5
	50	15	596.22	5	46.37	5
Average			162.82	5	12.51	5

bound procedure, we use a simple heuristic to obtain the initial lower bound. Specifically, we run the D2 method introduced in Glover et al. (1998) to obtain a starting complete solution whose objective function value serves as the initial lower bound in implementing the branch and bound method in all the experiments.

In our second experiment we test the efficiency of the offline computation of $z_{Unsel}(v)$ described in Section 4. We run the branch and bound method with and without this pre-calculation on the 10 *Glover*2 instances with $n = 50$ and on the 10 *Silva* instances with $n = 50$. Table 3 reports CPU times in seconds and the numbers of optimal solutions found in each case on each set of instances.

As shown in Table 3, the method without offline pre-calculation takes 162.82 s to implement, while the method with the pre-calculation only takes 12.51 s on average (and in both cases the method is able to obtain the optimal solution in the 20 examples considered). In this experiment we also compute the percentage of nodes in the search tree explored by the algorithm (i.e., we measure the effectiveness of the upper bound). The branch and bound algorithm only explores 0.09% of the nodes in the search tree, thus fathoming the rest of them. In the following experiments we consider the version with offline pre-calculation in our branch and bound algorithm.

In the third experiment, we compare three different versions of our branch and bound algorithm. The first version, BB, explores the search tree in ascending order of node number. In Section 4 we introduce $d_{\min}(v)$ and $d_{\max}(v)$ as the minimum and maximum potential contributions of a vertex v to any solution. The second version of the branch and bound, BBmax, explores the search tree in descending order of $d_{\max}$ value. Similarly, BBmin, explores the search tree in descending order of $d_{\min}$ value. Table 4 reports the average gap, the CPU time in seconds, and the number of optima that each method is able to match over all the *Glover*2 and *Silva* instances.

Table 4 shows that the BBmax method performs better than the other two variants considered. In particular, it shows that BB, BBmax and BBmin obtain 27, 34 and 31 optimal solutions (out of 50 *Glover*2 instances) achieved in 1689.9, 1291.3 and 1505.0 s, respectively. Similarly, in the 50 *Silva* instances, BB, BBmax and BBmin obtain 30, 30 and 30 optimal solutions achieved in 1760.3, 1714.0 and 1736.5 s, respectively. The application of BBmax leads to an average gap of 8.4% over the 100 instances considered in Table 4, while the application of BB and BBmin leads to average gaps of 9.8% and 8.8%, respectively. Therefore, we will use the BBmax variant in the following experiments in which we compare our branch and bound method with the linear integer formu-

Table 4
Results based on three versions of branch and bound algorithm.

	BB			BBmax			BBmin		
	Gap	CPU	#Opt	Gap	CPU	#Opt	Gap	CPU	#Opt
*Glover*2	6.4	1689.9	27	3.9	1291.3	34	4.8	1505.0	31
Silva	13.2	1760.3	30	12.9	1714.0	30	12.8	1736.5	30

Table 5
Results based on $BB_{\max}$ and F3.

Instance	n	m	BBmax			F3		
			Gap	CPU	#Opt	Gap	CPU	#Opt
*Glover*2	25	2	0.0	0.0	5	0.0	0.2	5
	25	7	0.0	0.0	5	0.0	15.0	5
	50	5	0.0	0.0	5	0.0	462.2	5
	50	15	0.0	0.4	5	27.7	3609.7	0
	100	10	0.0	4.4	5	71.5	3609.0	0
	100	30	8.6	3576.2	1	41.9	3603.8	0
	125	12	0.0	297.9	5	75.2	3600.0	0
	125	37	13.7	3600.0	0	45.3	3600.0	0
	150	15	5.4	1834.4	3	77.7	3600.0	0
	150	45	10.9	3600.1	0	52.7	3600.0	0
Silva	25	2	0.0	0.0	5	0.0	0.1	5
	25	7	0.0	0.0	5	0.0	4.5	5
	50	5	0.0	0.0	5	0.0	265.7	5
	50	15	0.0	22.8	5	25.8	3600.0	0
	100	10	0.0	38.3	5	71.1	3600.0	0
	100	30	31.7	3600.0	0	46.4	3600.0	0
	125	12	0.0	2678.9	5	76.1	3600.0	0
	125	37	34.6	3600.0	0	50.1	3600.0	0
	150	15	26.7	3600.1	0	78.1	3600.0	0
	150	45	35.6	3600.1	0	50.8	3600.0	0
Average			8.4	1502.7	3.2	39.5	2558.5	1.5

lation F3 solved with Cplex. As in the previous experiments, we limit the execution time on each instance to a maximum of 1 h (3600 s). Note that in the BBmax method, the total running time includes the offline pre-calculations.

Table 5 reports the results obtained with both methods, BBmax and F3, over the two sets of large instances, *Glover*2 and *Silva*. Each row displays the results for a group of five instances (with the same n and m values). As in the previous tables, we report the average gap, the CPU time in seconds, and the number of optima that each method is able to match. Note that some running times are a little longer than the time limit, especially in the CPU column under F3, since we check the running time when a complete iteration of the method is performed.

Table 5 shows that the Cplex solver with the F3 formulation is able to solve the medium-sized instances (up to $n = 50$ on both set of instances) within 1 h of CPU time. On the other hand, our branch and bound algorithm outperformers F3 since it is able to optimally solve all the medium-sized instances and some of the large instances ($n = 100$ and $m = 10$). Moreover, considering the 100 instances reported in Table 5, BBmax presents an average gap of 8.4% achieved in 1502.7 s, which compares favourably with the 39.5% obtained with F3 in 2558.5 s on average

6. Conclusions

We have developed an exact procedure based on the branch and bound method to provide solutions for the maximum diversity problem. We have introduced the partial solution as the set of solutions that share some vertices, and we have proposed several approaches to computing upper bounds on partial solutions. These bounds allow us to explore a relatively small portion of the nodes in the search tree when implementing our branch and bound procedure (0.09% on average).

We set out to investigate the efficiency of the offline pre-calculation and the order of node exploration. The empirical findings indicate that our method is able to solve medium-sized instances and obtains an average gap of 8.4% over all sets of instances. We have compared our method with the best approach in the existing literature, which is the linear integer formulation developed by Kuo et al. (1993). The results from a comparative study carried out with the well-known software Cplex favor the procedure that we proposed. However, it must be noted that neither of the two methods is able to solve large instances. Therefore, more research is necessary in this area. The focus of our future research will be on the development of tighter upper bounds and alternative strategies for examining the search tree to solve large MDP instances more efficiently.

Acknowledgement

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (TIN2006-02696) and by the Comunidad de Madrid – Universidad Rey Juan Carlos project (CCG08-URJC/TIC-3731).

References

Andrade, M., Andrade, P., Martins, S., Plastino, A., 2005. GRASP with path-relinking for the maximum diversity problem. In: Nikoletseas, S. (Ed.), Experimental and Efficient Algorithms, vol. 3503. Springer, Berlin, pp. 558–569.

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

Gallego, M., Duarte, A., Laguna, M., Martí, R., 2009. Hybrid heuristics for the maximum diversity problem. Computational Optimization and Applications. doi:10.1007/s10589-007-9161-6.

Ghosh, J.B., 1996. Computational aspects of the maximum diversity problem. Operations Research Letters 19, 175–181.

Glover, F., 1975. Improved linear integer programming formulations of nonlinear integer problems. Management Science 22, 455–460.

Glover, F., Woolsey, E., 1974. Converting the 0–1 polynomial programming problem to a 0–1 linear program. Operations Research 22, 180–182.

Glover, F., Kuo, C.C., Dhir, K.S., 1998. Heuristic algorithms for the maximum diversity problem. Journal of Information and Optimization Sciences 19, 109–132.

Katayama, K., Narihisa, H., 2004. An evolutionary approach for the maximum diversity problem. In: Hart, W., Krasnogor, N., Smith, J.E. (Eds.), Recent advances in memetic algorithms. Spinger, Berlin, pp. 31–47.

Kochenberger, G., Glover, F., 1999. Diversity Data Mining. Technical Report HCES-03-99. Hearin Center for Enterprise Science, University of Mississippi.

Kochenberger, G., Glover, F., 2006. A unified framework for modeling and solving combinatorial optimization problems: a tutorial. In: Hager, W., Huang, S.J., Pardalos, P.M., Prokopyev, O. (Eds.), Multiscale Optimization Methods and Applications. Springer, Berlin, pp. 101–124.

Kuo, C.C., Glover, F., Dhir, K.S., 1993. Analyzing and modeling the maximum diversity problem by zero-one programming. Decision Sciences 24, 1171–1185.

McConnell, S., 1988. The new battle over immigration. Fortune 117, 89–102.

Palubeckis, G., 2007. Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation 189, 371–383.

Porter, W.M., Rawal, K.M., Rachie, K.O., Wien, H.C., Williams, R.C., 1975. Cowpea Germplasm Catalog 1. International Institute of Tropical Agriculture, Ibadam, Nigeria.

Santos, L.F., Ribeiro, M.H., Plastino, A., Martins, S.L., 2005. A hybrid GRASP with data mining for the maximum diversity problem. In: Blesa, M., Blum, C., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics, vol. 3636. Springer, Berlin, pp. 116–127.

Silva, G.C., Ochi, L.S., Martins, S.L., 2004. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In: Ribeiro, C., Martins, C., Simone, L. (Eds.), Experimental and Efficient Algorithms, vol. 3059. Springer, Berlin, pp. 498–512.

Swierenga, R.P., 1977. Ethnicity in historical perspective. Social Science 52, 31–44.