An Exact Method for the Maximum Diversity Problem

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

MICAEL GALLEGO
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Micael.Gallego@urjc.es

ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

Abstract — In this article we first review previous integer formulations for the maximum diversity problem. This problem consists of selecting a subset of elements from a data 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 expressions for upper bounds for this problem. As far as we know, this is the first exact algorithm proposed for the maximum diversity problem. Empirical results with a collection of previously reported instances indicate that the proposed algorithm is able to solve medium size instances and compares favorably with the well-known optimizer Cplex when solving the previous formulations.

KeyWords: Branch and Bound, Integer Programming, Lower Bounds.

1. Introduction

The Maximum Diversity Problem (MDP) has been the subject of study since 1993 when Kuo, Glover and Dhir proposed different linear integer formulations. Since then, 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, where the goal of obtaining new varieties by controlled breeding stocks with desired qualities of diversity can be formulated as a MDP (Porter et al. 1975). The maximum diversity problem also appears in the context of ethnicity when it is studied from a historical perspective, as shown in Swierenga (1977), and more recently in the application of U.S. immigration policy that promotes of ethnic diversity among immigrants (McConnell 1988).

Ghosh (1996) proved the 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 these moves to be adapted to multiple settings. The authors compare the solution obtained with their heuristics with the optimal solution on small instances.

In the last few years, researchers have implemented different metaheuristics to obtain high quality solutions to medium and large MDP instances. Silva et al. (2004) present several GRASP algorithms for the MDP and test them on medium instances. Their extensive computational experiments show that their procedures outperform previous methods when they are allowed to run for extremely long times (their methods employ on average more than 20 hours of CPU time). Duarte and Martí (2006) proposed constructive and improvement algorithms for the MDP based on tabu search methodology and Gallego et al. (2006) implemented scatter search for the MDP. Both latter studies target large instances and, as shown in their computational testing, provide the best known solutions for this problem.

In this paper we first describe in Section 2 previous formulations for the MDP. Section 3 presents our theoretical contributions, which basically consist of upper bounds for partial solutions. A description of the exact algorithm is given in Section 4. It presents our for provably optimal solving method, which implements efficient strategies for branching and pruning. The paper finishes with a computational study and the associated conclusions.

2. Previous Formulations

Kuo, Glover and Dhir (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 with a MIP solver. The experimentation described in Section 5 compares these three formulations when solving small and medium MDP instances with Cplex 8.0.

The MDP consists of selecting a subset of elements from a data set 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 attributes. Let s_{ik} be the state or value of the k^{th} attribute of element i, where $k = 1, \ldots, K$. Then, the distance d_{ij} between elements i and j may be simply defined as the Euclidean distance (or as any other distance function). 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, $i = 0, .., n$.

$$\text{(F1)} \quad \text{Maximize} \quad z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

$$\text{Subject to} \quad \sum_{i=1}^{n} x_i = m$$

$$x_i = \{0, 1\} \qquad 1 \le i \le n$$

Glover and Wolsey (1974) show how to convert a 0-1 polynomial programming problem into a 0-1 linear programming problem. In particular, they propose replacing the product of variables $\prod_{j \in Q} x_j$ with a new variable x_Q, satisfying the following additional constraints:

$$\sum_{j \in Q} x_j - x_Q \le |Q| - 1$$

$$-x_j + x_Q \le 0, \ \forall j \in Q$$

$$x_Q \ge 0$$

If the original variable x_j is 0, the second additional constraint, $-x_j + x_Q \le 0$, implies $x_Q \le 0$; then, since $x_Q \ge 0$ (as given in the third additional constraint) we obtain $x_Q = 0$. Symmetrically, if the new variable x_Q is 0, the first additional constraint implies that any of the original variables must be 0, thus their product must also be 0. Then we have:

$$\prod_{j \in Q} x_j = 0 \quad \longleftrightarrow \quad x_Q = 0$$

On the other hand, if all the original variables have value 1, the first additional constraint results in $-x_Q \le -1$ and, together with any of the second additional constraints, $x_Q \le 1$, leads to $x_Q = 1$. Symmetrically, if x_Q takes the value 1, the second additional constraints force the original variables to be larger than or equal to 1; therefore, they must be 1 (since they are binary variables). Therefore, $\prod_{j \in Q} x_j = x_Q$.

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 in which the additional constraints have been added.

$$
\begin{array}{lll}
\text{(F2)} \quad \text{Maximize} & z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} y_{ij} & \\
\quad\quad\quad \text{Subject to} & \sum_{i=1}^{n} x_i = m & \\
 & x_i + x_j - y_{ij} \le 1 & 1 \le i < j \le n \\
 & -x_i + y_{ij} \le 0 & 1 \le i < j \le n \\
 & -x_j + y_{ij} \le 0 & 1 \le i < j \le n \\
 & y_{ij} \ge 0 & 1 \le i < j \le n \\
 & x_i = \{0, 1\} & 1 \le i \le n
\end{array}
$$

Glover (1975) introduces some inequalities to handle quadratic integer programs with both real and binary variables. Consider a variable *w* and a 0-1 variable *x* such that:

$$U_0 \ge w \ge L_0 \text{ if } x = 0 \text{ , and } \quad U_1 \ge w \ge L_1 \text{ if } x = 1.$$

A simple way to model this situation is given by:

$$U_0 + (U_1 - U_0)x \ge w \ge L_0 + (L_1 - L_0)x$$

U_0, U_1, L_0, L_1 are usually non-constant values and then the expressions $(U_1\text{-}U_0)x$ and $(L_1\text{-}L_0)x$ will usually be nonlinear. Then, Glover (1975) introduces the following upper and lower bounds:

$$\overline{U}_0 \ge U_0 \ge \underline{U}_0, \ \overline{L}_0 \ge L_0 \ge \underline{L}_0, \ \overline{U}_1 \ge U_1 \ge \underline{U}_1, \ \overline{L}_1 \ge L_1 \ge \underline{L}_1$$

to rewrite the inequalities above with these new expressions:

$$U_0 + (\overline{U}_1 - \underline{U}_0)x \ge w \ge L_0 + (\underline{L}_1 - \overline{L}_0)x,$$
$$U_1 + (\overline{U}_0 - \underline{U}_1)(1-x) \ge w \ge L_1 + (\underline{L}_0 - \overline{L}_1)(1-x).$$

Kuo et al. (1993) apply this transformation to F1 by decomposing the objective function into the sum of *w*-values:

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

Defining the following bounds,

$$\overline{U}_0 = \underline{U}_0 = \overline{L}_0 = \underline{L}_0 = 0 \quad , \quad \overline{U}_1 = \overline{L}_1 = \overline{D}_i = \sum_{j=i+1}^{n} \max(0, d_{ij})$$

$$\text{and} \quad \underline{U}_1 = \underline{L}_1 = \underline{D}_i = \sum_{j=i+1}^{n} \min(0, d_{ij}),$$

F1 is rewritten as F3:

(F3) Maximize: $z = \sum_{i=1}^{n-1} w_i$

Subject to: $\sum_{i=1}^{n} x_i = m$

$$-\overline{D}_i \cdot x_i + w_i \leq 0, \qquad 1 \leq i \leq n \quad (1)$$

$$-\sum_{j=i+1}^{n} d_{ij} x_j + \underline{D}_i \cdot (1 - x_i) + w_i \leq 0, \qquad 1 \leq i \leq n \quad (2)$$

$$x_i = \{0, 1\} \qquad 1 \leq i \leq n \qquad (3)$$

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

$$-\sum_{j=i+1}^{n} d_{ij} x_j + \underline{D}_i + w_i \leq 0 \longrightarrow w_i \leq \sum_{j=i+1}^{n} d_{ij} x_j - \underline{D}_i$$

and by definition:

$$\underline{D}_i \leq \sum_{j=i+1}^{n} d_{ij} x_j$$

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 their respective contribution 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 equivalents. Therefore, the maximum diversity problem can be formulated as F1, F2 and F3. In the computational experiments reported in Section 6 we will see their effectiveness when solving a set of instances.

3. Upper Bounds for Partial Solutions

The maximum diversity problem can be defined in terms of graphs. We can represent each element in the problem as a vertex in a graph and the distance between elements as the weight or cost associated to the corresponding edge. The MDP then consists of selecting a subset of vertices in such a way that the sum of the distances between them is maximized. In this section we provide some expressions to compute the upper bound of the MDP value of a set of solutions that share some vertices, which will be called a *partial solution.*

Let $V=\{v_1, v_2, \ldots, v_n\}$ 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={$s_1, s_2, \ldots, s_k$} with $k < m$ elements as a subset of V ($Sel \subset V$). Let S_{Sel} be the set of solutions of the MDP in which all the elements in *Sel* are selected.

Given a 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 value z of 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).

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

We use the example given in Figure 1 and the associated Euclidean distance matrix in Table 1 to illustrate these concepts. The dimensions of the example problem are $n = 6$ and $m = 4$.

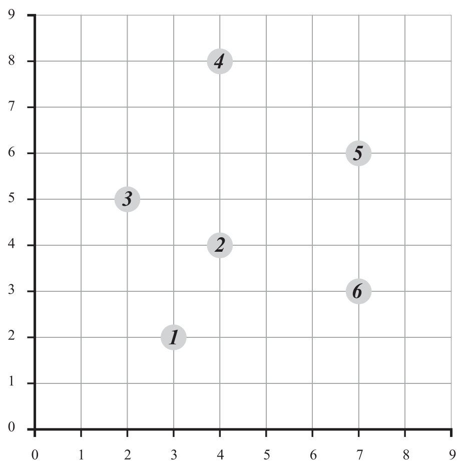

Figure 1. MDP example

	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	-

Table 1: Distance matrix

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 solution x={1,3,5,6} in S_{Sel}, then *Unsel*(x)={5,6}. We can compute its value as $z = z_1 + z_2 + z_3 = 3.16 + 20.27 + 3.00 = 26.43$ where

$z_1 = d(1,3) = 3.16$
$z_2 = d(1,5) + d(3,5) + d(1,6) + d(3,6) = 5.66 + 5.10 + 4.12 + 5.39 = 20.27$
$z_3 = d(5,6) = 3.00$

Figure 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 value of any solution in S_{Sel}. Propositions 1 and 2 provide an upper bound for the value of z_2 and z_3 respectively for any solution in S_{Sel}. Proposition 3 provides an improved upper bound for the value of $z_2 + z_3$.

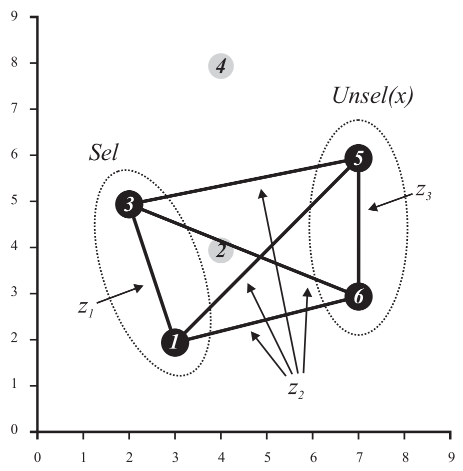

Figure 2. Decomposition of the solution value

Given a set of vertices, *Sel*={s_1, s_2,…, s_k}, and a vertex $v \in V\text{-}Sel$, 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 *Sel*={s_1, s_2, …, s_k} with $k < m$, let $z_{Sel}^1, z_{Sel}^2, ..., z_{Sel}^{n-k}$ be the ordered values of $z_{Sel}(v)$ for all v in *V-Sel* (where z_{Sel}^1 is the maximum); then, for any solution x in S_{Sel}:

$$UB_2 = \sum_{j=1}^{m-k} z_{Sel}^j \geq z_2 \; .$$

Proof

Given the solution x={s_1, s_2,…, s_k, u_1, u_2,…, u_{m-k}} in S_{Sel}, it is clear that:

$$z_2 = \sum_{i=1}^{k}\sum_{j=1}^{m-k} d(s_i, u_j) = \sum_{j=1}^{m-k} z_{Sel}(u_j) \leq \sum_{j=1}^{m-k} z_{Sel}^j \qquad \blacksquare$$

Given a partial solution *Sel*={s_1, s_2,…, s_k} with $k < m$, let $d_{Unsel}^1(v)$, $d_{Unsel}^2(v)$,…, $d_{Unsel}^{n-k-1}(v)$ be the ordered values of the distances between v in *V-Sel* and the other vertices in *V-Sel* (where $d_{Unsel}^1(v)$ is the maximum). We define $z_{Unsel}(v)$ as:

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

we can interpret $z_{Unsel}(v)$ as an upper bound of 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 Figure 1 and Table 1 with *Sel*={1,3}, if we consider vertex 2, we have *d*(2,4)=4.00, *d*(2,5)=3.61 and *d*(2,6)=3.16. Then,

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

Proposition 2

Given a partial solution *Sel*={s_1, s_2, …, s_k} with $k < m$, let $z^1_{Unsel}, z^2_{Unsel},\dots,z^{n-k}_{Unsel}$ be the ordered values of $z_{Unsel}(v)$ for all *v* in *V-Sel* (where z^1_{Unsel} is the maximum); then, for any solution *x* in S_{Sel}:

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

Proof

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

$$z_3=\sum_{i=1}^{m-k-1}\sum_{j=i+1}^{m-k} d(u_i,u_j)=\frac{1}{2}\sum_{i=1}^{m-k}\sum_{j=1,j\neq i}^{m-k} d(u_i,u_j)\leq \sum_{i=1}^{m-k} z_{Unsel}(u_i)\leq \sum_{i=1}^{m-k} z^i_{Unsel} \quad \blacksquare$$

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:

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

It is easy to check that 29.44 is larger than any solution value in this set:

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

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, …, s_k} with $k < m$ for any *v* in *V-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, …, s_k} with $k < m$, let $z^1, z^2,\dots,z^{n-k}$ be the ordered values of *z*(*v*) for all *v* in *V-Sel* (where z^1 is the maximum); then, for any solution *x* in S_{Sel}:

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

Proof

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

$$z_2 + z_3 = \sum_{j=1}^{k}\sum_{i=1}^{m-k} d(s_j,u_i) + \sum_{i=1}^{m-k-1}\sum_{j=i+1}^{m-k} d(u_i,u_j) = \sum_{j=1}^{k}\sum_{i=1}^{m-k} d(s_j,u_i) + \frac{1}{2}\sum_{i=1}^{m-k}\sum_{j=1,j\neq i}^{m-k} d(u_i,u_j)$$

$$z_2 + z_3 \le \sum_{i=1}^{m-k} z_{Sel}(u_i) + \sum_{i=1}^{m-k} z_{Unsel}(u_i) = \sum_{i=1}^{m-k} z\ (u_i) \le \sum_{i=1}^{m-k} z^i = UB_{23}$$

■

In the example above we have

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

Similarly,

$$z(4) = 9.96 + \max(\frac{4.00}{2},\frac{3.61}{2},\frac{5.83}{2}) = 12.61$$
$$z(5) = 10.76 + \max(\frac{3.61}{2},\frac{3.61}{2},\frac{3.00}{2}) = 12.57$$
$$z(6) = 9.51 + \max(\frac{3.16}{2},\frac{5.83}{2},\frac{3.00}{2}) = 12.43$$

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 $UB_{23} \le UB_2+UB_3$ because UB_2 is computed with respect to the maximum $z_{Sel}(v)$ values and UB_3 is computed with respect to the maximum $z_{Unsel}(v)$ values. Then, the vertices v in UB_2 and UB_3 may be different, while UB_{23} is computed with respect to the vertices for which $z_{Sel}(v)+z_{Unsel}(v)$ is maximum. We use the same vertices v 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.

4. Search Tree

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 solution, the search tree 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 *leaf* or final nodes that represent complete solutions).

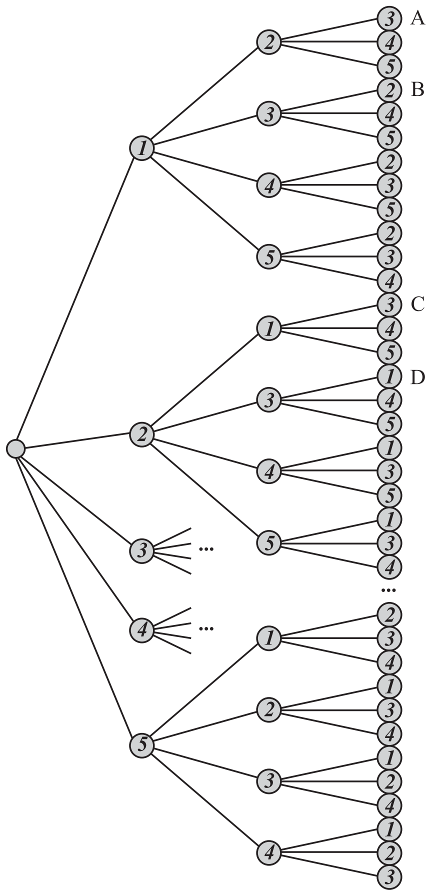

Figure 3. Basic search tree

In a first approach to this problem we can 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,.., *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 more vertex. Figure 3 represents this basic search tree for an example with *n* = 5 and *m* = 3.

As shown in Figure 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 solutions of the MDP. So 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*, 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 value of their label. In general terms node *i* in level *k* branches into *n*-(*m*-*k*)-*i* nodes, beginning with node *i*+1 and ending with node *n-(m-k)*. In the example above with *n* = 5 and *m* = 3, Figure 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 Figure 4 with that in Figure 3, we can see that the size has been substantially reduced by avoiding the repetitions.

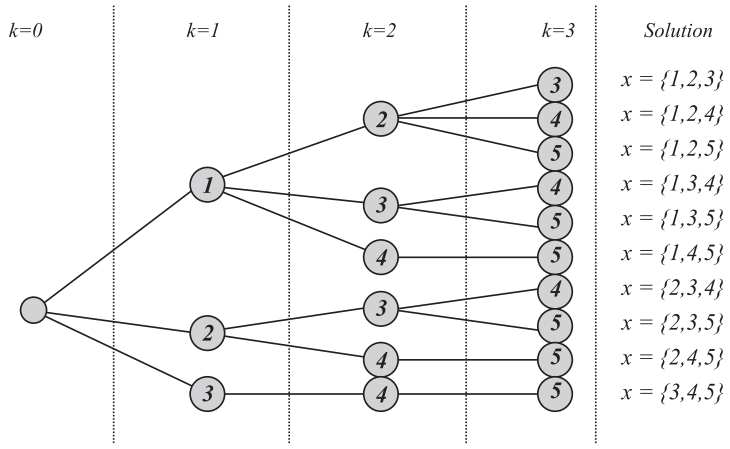

Figure 4. Search tree

A direct way to implement the construction of the search tree is by means of a recursive procedure. Figure 5 shows the pseudo-code of this procedure. This algorithm has two input parameters, level k and the partial *solution* under construction. Once the final level is reached and the solution is completed, the method prints it and performs a backward step. For the sake of simplicity, this figure does not show the implementation details and data structures that make our final procedure more efficient.

```
procedure createSolutions (k , solution)
var
  init_value, i: integer;
begin
  if k = m+1 then
    print(solution);                    // Solution output
  else
    if k = 1 then                       // Initialization for each level
      init_value := 1;
    else
      init_value := solution[k - 1] + 1;
    end if
    for i := init_value to n-(m-k) do   // Create the new nodes in the tree
      solution[k]:=i;
      createSolutions(k+1, solution);   // Recursive call
    end for
  end if
end
```

Figure 5. Search tree construction procedure

The value of a heuristic solution gives a lower bound for the problem. The tree is explored in a depth first search where nodes represent partial solutions (except the *leaf* nodes in level m that represent complete solutions). The upper bound z_1+UB_{23} is computed at each node. If it is lower than the lower bound we fathom the node, otherwise we branch the node and explore its first *child* node. We apply a second test to fathom nodes in the search tree based on the following proposition.

Given a solution $x=\{s_1, s_2, \dots, s_m\}$, we can express its value as:

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

represents the contribution of s_i to the value z. From this expression we can introduce the maximum, $d_{\max}(v)$, and minimum, $d_{\min}(v)$, potential contribution of a vertex v to any solution. Let $d(v,v_{\sigma(1)})$, $d(v,v_{\sigma(2)}),\dots$, $d(v,v_{\sigma(n-1)})$ be the ordered values of the distances between v and the other vertices in the graph (where $d(v,v_{\sigma(1)})$ is the maximum).

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

It is clear from the definition that any solution with vertex v selected verifies:

$$d_{\min}(v) \le d(v) \le d_{\max}(v)$$

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

Consider that u is in the optimal solution $x=\{u, s_2, s_3, \dots, s_m\}$ and we will see that it leads us to a contradiction. The value z of x can be broken down into:

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

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

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

Similarly, if we consider the solution $y=\{v, s_2, s_3, \dots, s_m\}$, its value z' verifies:

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

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.

■

Figure 6 illustrates the application of Proposition 4 to fathom some nodes in the search tree in an example with n=6 and m=3. Consider in this example that $d_{\max}(3)<d_{\min}(1)$, then according to Proposition 4, if vertex 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. Figure 6 shows the search tree of this example in which ten 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.

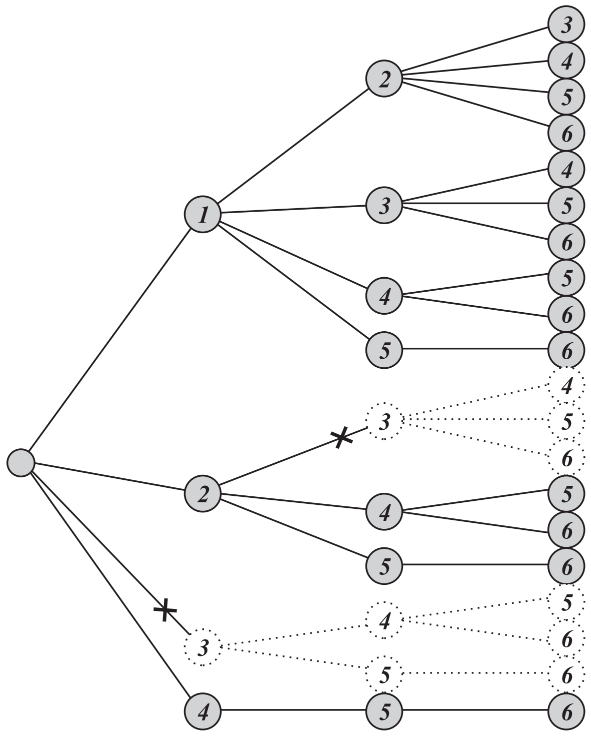

Figure 6. Fathomed nodes in search tree

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 any v in V-Sel, where $z_{Unsel}(v)$ involves several time-consuming calculations. However, we have found that the way in which we generate the search tree, in order 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 of the selected vertices (*max_label*). For example, in the search tree shown in Figure 6, 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 Figure 6. Therefore, we compute the values of $z_{Unsel}(v)$ for each possible combination of k and *max_label* at the beginning of the method. In the next section we empirically show the effectiveness of this offline computation.

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 comparing it with the solution obtained with the previous linear integer formulations. We have implemented the branch and bound 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 is an extension of the first one to target large graphs, and the third set is from Silva et al. (2004):

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 to 100 range. The number of coordinates for each point is also randomly generated between 2 and 21. Glover, Kuo and Dhir (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.
*Glover*2:	This data set consists of 50 matrices constructed with the same graph generator employed in the *Glover* set. We generated ten instances with n=25, 50, 100, 125 and 150. For each value of n we consider m=0.1n, 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, Ochi and Martins (2004). We use their generator to construct instances of the same size as those in the *Glover*2 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 hour (3,600 seconds) of computer time. Table 2 shows, for each combination of the n and m values in the *Glover* set of instances, the average value of the gap, Gap, the CPU time in seconds, CPU, and the number of optima that each method is able to match, #Opt. The value of the gap is computed as the upper bound minus the lower bound and divided by the upper bound (and multiplied by 100).

Table 2 shows that, as expected, formulation F3 produces better results than F1 and F2 in shorter running time. Specifically, F1 presents an average gap value of 0.7% achieved in 591.6 seconds on average, F2 0.8% achieved in 711.7 seconds, and F3 0.0% achieved in 96.9 seconds. On the other hand, F1, F2 and F3 obtain 65, 66 and 75 optimal solutions respectively (out of 75 instances).

		F1			F2			F3		
n	*m*	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
Summary		0.7	591.6	65	0.8	711.7	66	0.0	96.9	75

Table 2. Linear integer formulations

In the following experiments we test our branch and bound method. In line with testing the effectiveness of the branch and bound procedure, we use a simple heuristic to obtain the initial lower bound. Specifically, we run the D_2 method introduced in Glover et al. (1998) to obtain the initial solution (its value is used as the initial lower bound) of 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 ten *Glover*2 and ten *Silva* instances with n=50. We do not reproduce the table of this experiment, but we report that the method without offline pre-calculation takes 574.21 seconds, while the method with the pre-calculation only takes 17.12 seconds on average (and in both cases the method is able to obtain the optimal solution in the twenty examples considered). In this experiment we also compute the percentage of nodes in the search tree explored with the algorithm (i.e. we measure the effectiveness of the upper bound). The branch and bound 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 lexicographical order. In Section 4 we introduce $d_{\min}(v)$ and $d_{\max}(v)$ as the minimum and maximum potential contribution of a vertex v to any solution. The second version of the branch and bound, BBmax, explores the search tree in the order given by $d_{\max}(v)$. In particular, we order the vertices according to their $d_{\max}$-value (where the vertex with the maximum value is the first), and the method explores the nodes in the search tree according to this order. Similarly, BBmin, explores the search tree in the order given by $d_{\min}(v)$ (visiting first the nodes with maximum $d_{\min}$-value of their vertices). Table 3 reports the average value of the gap, Gap, the CPU time in seconds, CPU, and the number of optima that each method is able to match, #Opt, for the fifty *Glover*2 and fifty *Silva* instances

	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 3. Comparison of branch and bound variants

Table 3 shows that the BBmax method performs better than the other two variants considered. This table 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 seconds 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 seconds respectively. Regarding the Gap values, BBmax obtains 8.4% on average over the 100 instances considered in Table 3, which compares favorably with 9.8% of the BB and with 8.8% of the BBmin. Therefore, we will use the BBmax variant in the following experiments in which we compare our branch and bound method with the linear integer formulation F3 solved with Cplex. As in the previous experiments, we limit the execution time on each instance to a maximum of 1 hour (3,600 seconds).

			BBmax			F3		
Instance	*n*	*m*	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
Summary			8.4	1502.7	64	39.5	2558.5	30

Table 4. Comparison between best solution methods

Table 4 reports the results obtained with both methods, BBmax and F3, over the two sets of large instances, *Glover*2 and *Silva*. Each row reports the results on the group of five instances (with the same *n* and *m* values). As in previous tables, we report the average value of the gap, Gap, the CPU time in seconds, CPU, and the number of optima that each method is able to match, #Opt.

Table 4 shows that the Cplex solver with the F3 formulation is able to solve the medium size instances (up to $n = 50$ on both set of instances) within 1 hour of CPU time. On the other hand, our branch and bound algorithm outperforms the other method since it is able to optimally solve all the medium instances ($n = 50$) and some of the large size instances ($n = 100$, $m = 10$). Moreover, considering the 100 instances reported in Table 4, BBmax presents an average gap value of 8.4% achieved in 1502.7 seconds, which compares favourably with the 39.5% obtained with F3 in 2558.5 seconds on average.

6. Conclusions

We have developed an exact procedure based on the branch and bound methodology to provide bounds and solutions for the maximum diversity problem. As far as we know, this is the first exact algorithm proposed for this problem. We have introduced the partial solution as the set of solutions that share some vertices, and we have proposed several theoretical results to compute upper bounds on partial solutions. These bounds allow us to explore a relatively small portion of the nodes in the search tree of the branch and bound procedure (0.09% on average).

We perform extensive preliminary experimentation to study the effect of the elements proposed in the solution method, such as the offline pre-calculation of time-consuming computations in the upper bounds and the order of node exploration. The experimentation shows that our method is able to solve medium size instances (up to $n = 100$) and obtains an average gap, over all sets of instances, of 8.4%. We have compared our method with the best previous linear integer formulation (Kuo et al., 1993) solved with the well-known software Cplex. The comparison favors the proposed procedure.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Refs. TIN2005-08943-C02-02, TIN2006-02696) and by the Comunidad de Madrid – Universidad Rey Juan Carlos project (Ref. URJC-CM-2006-CET-0603).

References

Campos, V., F. Glover, M. Laguna and R. Martí (2001) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," *Journal of Global Optimization*, vol. 21, pp. 397-414.

Duarte, A. and R. Martí (2006) “Tabu Search and GRASP for the Maximum Diversity Problem,” to appear in the *European Journal of Operational Research*.

Gallego, M., Duarte, A., Laguna, M. and Martí, M. (2006), "Hybrid Heuristics for the Maximum Diversity Problem", Technical report, University of Valencia, Spain.

Ghosh, J. B. (1996) “Computational Aspects of the Maximum Diversity Problem,” *Operations Research Letters*, vol. 19, pp. 175-181.

Glover, F. (1975) "Improve Linear Integer Programming Formulations of Nonlinear Integer Problems" *Management Science* 22 (4), 455 - 460.

Glover, F. and Wolsey, R.E. (1974) "Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Programming" *Operations Research* 22 (1), 180-182.

Glover, F., C. C. Kuo, and K. S. Dhir (1998) “Heuristic Algorithms for the Maximum Diversity Problem,” *Journal of Information and Optimization Sciences*, vol. 19, no. 1, pp. 109-132.

Glover, F., M. Laguna (1997) *Tabu Search*. Kluwer Academic Publisher.

Kuo, C. C., F. Glover and K. S. Dhir (1993) “Analyzing and Modeling the Maximum Diversity Problem by Zero-One Programming,” *Decision Sciences*, vol. 24, no. 6, pp. 1171-1185.

Laguna, M. and R. Martí (2003) *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers: Boston.

Silva, G. C., L. S. Ochi and S. L. Martins (2004) “Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem,” *Lecture Notes in Computer Science* 3059, Springer: Berlin Heidelberg, pp. 498-512.