

A simple and effective algorithm for the MaxMin diversity problem

Daniel Cosmin Porumbel · Jin-Kao Hao · Fred Glover

Published online: 15 May 2011
© Springer Science+Business Media, LLC 2011

Abstract The challenge of maximizing the diversity of a collection of points arises in a variety of settings, including the setting of search methods for hard optimization problems. One version of this problem, called the Maximum Diversity Problem (MDP), produces a quadratic binary optimization problem subject to a cardinality constraint, and has been the subject of numerous studies. This study is focused on the Maximum Minimum Diversity Problem (MMDP) but we also introduce a new formulation using MDP as a secondary objective. We propose a fast local search based on separate add and drop operations and on simple tabu mechanisms. Compared to previous local search approaches, the complexity of searching for the best move at each iteration is reduced from quadratic to linear; only certain streamlining calculations might (rarely) require quadratic time per iteration. Furthermore, the strong tabu rules of the drop strategy ensure a powerful diversification capacity. Despite its simplicity, the approach proves superior to most of the more advanced methods from the literature, yielding optimally-proved solutions for many problems in a matter of seconds and even attaining a new lower bound.

Keywords Maximum diversity · MaxMin diversity · Tabu search

1 Introduction

Let Z be a finite collection of points (elements), and let $Z(k) = \{X \subset Z : |X| = k\}$, the set of all k element subsets of Z , where $2 \leq k \leq |Z| - 1$. Associated with each pair of points

D.C. Porumbel
UARtois, LGI2A, Univ. Lille-Nord de France, Technoparc Futura, 62400 Béthune, France
e-mail: daniel.porumbel@univ-artois.fr

J.-K. Hao (✉)
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
e-mail: hao@info.univ-angers.fr

F. Glover
OptTek Systems, Inc., 4421 17th Street, Boulder, CO 80302, USA
e-mail: glover@opttek.com

$x, y \in Z$ is a real number $d(x, y)$ ($= d(y, x)$) called a *distance*. (In our formulation, $d(x, y)$ does not have to satisfy the properties of a customary distance metric and may even be negative.) For simplicity, when referring to $d(x, y)$ we understand that $x \neq y$. The classical Maximum Diversity Problem (MDP) requires identifying a set $X^* \in Z(k)$ that maximizes the sum of the distances $d(x, y)$ over all distinct pairs $x, y \in X^*$. More precisely, the problem may be expressed as:

$$\text{MDP: Find } X^* = \arg \max \left(\sum_{x, y \in X} d(x, y) : X \in Z(k) \right).$$

A number of computational studies of MDP have been performed, including those of Kuo et al. (1993), Ghosh (1996), Glover et al. (1998), Silva et al. (2004), Andrade et al. (2005), Duarte and Marti (2007), Gallego et al. (2009), Palubeckis (2007), Aringhieri et al. (2008), Santos et al. (2008), Wang et al. (2009) and Aringhieri and Cordone (2011). As observed in Kuo et al. (1993) the maximum diversity problem has applications in plant breeding, social problems, ecological preservation, pollution control, product design, capital investment, workforce management, curriculum design and genetic engineering.

In certain contexts, however, a more useful definition of diversity involves the goal of finding a set X^* in $Z(k)$ that maximizes the minimum distance between the points $x, y \in X^*$ instead of the sum of the distances between these points. This “MaxMin” form of diversity has application to achieving diversification goals for metaheuristics, as noted in Glover and Laguna (1997), and is relevant to the area of simulation optimization (see, e.g., April et al. 2003). To capture this form of diversity, the classical definition of the MaxMin Diversity Problem (MMDP) may be expressed as:

$$\text{MMDP: Find } X^* = \arg \max \left(\text{Min}_{x, y \in X} (d(x, y)) : X \in Z(k) \right).$$

Methods for the MMDP problem, both heuristic and exact, have been proposed and investigated by several authors, including Erkut (1990), Kincaid (1992) and Ghosh (1996) and Della Croce et al. (2009). A comprehensive survey of previous work can be found in Resende et al. (2010).

In terms of practical results for MMDP, there are two very recent studies that achieved very high performances, thus providing a solid comparison base for benchmarking new algorithms. Resende et al. (2010) presented a refined GRASP (with Path Relinking) approach, as well as extensive comparisons with previous algorithms (e.g. Erkut 1990; Kincaid 1992; Ghosh 1996). Della Croce et al. (2009) described an algorithm which uses a powerful clique heuristic (i.e., Grosso et al. 2008) as an internal component in a dichotomic search. This algorithm reached very competitive bounds—including several that the authors proved optimal using an exact clique solver Östergård (2002).

The objective of this paper is to present a simple and effective approach to diversity problems, using a new MMDP formulation based on both MMDP and MDP (Sect. 2.1). To deal with this problem, we first apply a constructive heuristic (Sect. 2.2), and then we place a special emphasis on a new tabu search algorithm based on *separate* drop and add operations (Sect. 2.3). The proposed search approach aims at meeting two essential challenges in heuristic algorithm design. First, in order to render the search process as fast as possible, the computational complexity of the neighborhood exploration is reduced to linear and all calculations are effectively streamlined. Secondly, the search process can not get stuck by keeping certain elements of a solution assignment in the selected set X for an indefinite duration, because each selected element is systematically dropped after a determined number of iterations.

The paper is organized as follows. Section 2 is devoted to a detailed presentation of the constructive stage and of the new search algorithm based on drop and add moves. Numerical results are presented in Sect. 3, followed by conclusions in Sect. 4. The Appendix provides detailed instance-by-instance comparisons between our approach and state-of-the-art algorithms.

2 The proposed approach

The main component of our approach is a simple tabu search algorithm that is very fast (the iteration complexity is reduced to minimum) and that exhibits strong diversification qualities. While several search methods for MMDP or MDP are already available in the literature (see Introduction), they typically use a neighborhood of quadratic cost based on swap moves, i.e., a neighbor is obtained by replacing a selected element with a non-selected one. A novelty of our search strategy is that such swap moves are executed as a succession of two separate steps: drop and add. These two steps have linear complexity; furthermore, all calculations are effectively streamlined, and so, the total iteration computational cost becomes considerably less expensive than in other approaches. The drop operation always removes the “oldest” selected element: each selected point is replaced with a different one after exactly k iterations. This simple policy, which represents an elementary instance of a short-term tabu search rule, proves very effective in helping the search process to avoid looping and ensure diversity.

2.1 Problem formulation

We are interested in going beyond the classical formulation of MMDP to give a more general formulation that includes the objective of MDP as a secondary objective, i.e., we seek first to maximize the minimum distance between points in the chosen set X^* and subject to this also seek to maximize the sum of the distances between these points. The inclusion of this secondary objective is motivated by the fact that there may be a number of solutions that qualify as optimal when the MMDP objective is considered solely by itself, and therefore it is useful to have a meaningful way to differentiate among these “tied optimal” solutions, particularly by means of a criterion such as that of MDP which has also been found of interest in the literature. This formulation is also motivated by observations of Glover et al. (1998) that it can be important to differentiate among solutions that are equally valued by reference to the classical objective.

Formally, define $Z^o(k) = \{X^o = \arg \max(\text{Min}_{x,y} \in X(d(x, y) : X \in Z(k))\}$. Then we may write the more general form of MMDP, denoted MMDP^o as

$$\text{MMDP}^o: \text{ Find } X^* = \arg \max \left(\sum_{x,y \in X^o} d(x, y) : X^o \in Z^o(k) \right). \tag{1}$$

Alternatively, we may also state MMDP^o by referring to a small positive number ε (chosen so that the maximization of the summed distances is subordinate to the maximization of the min distance) and writing

$$X^* = \arg \max \left(\text{Min}_{x,y \in X} \left(d(x, y) + \varepsilon \sum_{x,y \in X} d(x, y) : X \in Z(k) \right) \right). \tag{2}$$

We will call the objective function of MMDP^o, variously expressed in the form of (1) or (2) as the MMDP^o criterion. Clearly a solution that is optimal for MMDP^o will also be optimal for MMDP, though the converse is not true.

2.2 Constructive procedure for MMDP^o

The first stage (see Algorithm 1 below) of our framework consists of an elementary constructive algorithm to create a set X in $Z(k)$ that constitutes a first candidate for an optimal solution.

Algorithm 1 (Constructive Algorithm)

Choose an initial point $x^* \in Z$ and set $X(1) = \{x^*\}$.

Set $h = 1$.

While $h < k$:

Set $h := h + 1$ and choose a point $x^* \in Z - X(h - 1)$ such that

$$x^* = \arg \max(\text{Min}(d(x, y) : x \in Z - X(h - 1), y \in X(h - 1))),$$

where ties for x^* are broken by selecting a point that maximizes $\sum d(x, y)$.

Set $X(h) = X(h - 1) \cup \{x^*\}$.

Endwhile

Identify the first candidate for an optimal MMDP^o solution by $X^* = X(k)$.

The initial point x^* to include in $X(1) = \{x^*\}$ is obtained by selecting a point x^* that maximizes the sum of distances towards the other points. The algorithm, which depends importantly on this initial x^* , is a greedy algorithm for passing from $X(h - 1)$ to $X(h)$ by reference to the MMDP objective, subject to breaking ties relative to the MDP objective.

The set $X^* = X(k)$ produced at the final stage of this constructive stage is then modified by the Drop-Add search described next.

2.3 Simple tabu search algorithm based on drop and add moves

Both the constructive and the search algorithms make use of an iteration counter, denoted by iter , that progresses from 1 through k as the successive steps of the Constructive Algorithm are executed, and continues to be incremented by 1 at each subsequent iteration of the Drop-Add Simple Tabu Search Algorithm that follows. For each point x that belongs to a current set $X(k)$, we let $\text{iter}(x)$ denote the iteration at which x was added to $X(k)$. The local search also drops elements from $X(k)$, and for each x that is dropped, we likewise let $\text{iter}(x)$ identify the iteration at which x was removed from $X(k)$. (Initially, $\text{iter}(x) = 0$ for all $x \in Z$.) Hence, more precisely, $\text{iter}(x)$ denotes the latest iteration at which x was either added to or dropped from $X(k)$.

We refer to a subset $\text{Add } X(k)$ of the current set of points $Z - X(k)$ that consists of *eligible add* points, that is, those eligible to be considered for being added to $X(k)$. We define $\text{Add } X(k)$ as $Z - X(k)$, and so, all points outside of $X(k)$ are eligible to be added. Similarly, we also refer to a subset of eligible drop points $\text{Drop } X(k)$: this set employs an exceedingly simple tabu rule by always containing only the “oldest” point in $X(k)$, i.e., the point $x^\# = \arg \min(\text{iter}(x) : x \in X(k))$. By this definition, the value of $\text{iter}(x^\#)$ can be identified as $\text{iter}(x^\#) = \text{iter} - (k - 1)$, where iter denotes the current value of the iteration counter. Hence, at the end of the Constructive Algorithm, $\text{iter} = k$ and $\text{iter}(x^\#) = 1$, identifying $x^\#$ as the first point added to $X(k)$.

Algorithm 2 below presents the algorithmic template of the proposed local search. The first step simply drops the “oldest selected point”, i.e., the selected point $x^\# \in X(k)$ that verifies $\text{iter}(x^\#) = \text{iter} - (k - 1)$. This operation temporarily produces a set $X(k - 1)$; afterwards, Step 2 chooses the point x^* from $\text{Add } X(k)$ by the same criterion used in the Constructive Algorithm, i.e., x^* is chosen to maximize the minimum distance from x^* to any point in $X(k - 1)$, and, subject to this, to maximize the sum of the distances to the points in $X(k - 1)$. Step 3 updates the value of X^* if $X(k)$ is better than the best visited solution so far. Finally, $\text{iter}(x^\#)$ and $\text{iter}(x^*)$ are updated; the iteration counter is incremented in the last instruction.

Algorithm 2 (Drop-Add Simple Tabu Search)

While a termination condition is not satisfied:

1. From the current set $X(k)$, select the point $x^\# \in \text{Drop } X(k)$ to produce the set $X(k - 1) = X(k) - \{x^\#\}$.
2. Choose a point $x^* \in \text{Add } X(k) = Z - X(k)$ such that $x^* = \arg \max(\text{Min}(d(x, y) : x \in \text{Add } X(k), y \in X(k - 1)))$, breaking ties for x^* by selecting a point that maximizes $(\sum d(x, y) : y \in X(k - 1))$.
Set $X(k) = X(k - 1) \cup \{x^*\}$.
3. **If** $X(k)$ improves on the best set X^* by the MMDP⁰ criterion **Then**
Record $X(k)$ as the new X^*
- End If**
4. Set $\text{iter}(x^\#) = \text{iter}(x^*) = \text{iter}$; Set $\text{iter} := \text{iter} + 1$.

End While

The method stops when no improvement according to the MMDP⁰ criterion is made in X^* for a number of iterations, denoted as maxNoGain in Sect. 3.

2.4 Streamlining the calculations

An important objective of the proposed search strategy is to avoid performing iterations of quadratic complexity. Since Step 2 of Algorithm 2 requires going through all points of $\text{Add } X(k) = Z - X$, it is important to be able to access certain information in $O(1)$ —e.g., the minimum distance from each point $x \in Z - X$ to points from X (denoted by $\text{MinDist}(x)$ below). For this purpose, we maintain and update the following three records for each $x \in Z$ after each add or drop operation:

- $\text{MinDist}(x) = \text{Min}\{d(x, y) : y \in X\}$,
- $\text{SumDist}(x) = (\sum d(x, y) : y \in X)$,
- $\text{MinDistCount}(x) = |\{y \in X : d(x, y) = \text{minDist}(x)\}|$, i.e., the number of elements having x as the closest point.

Each of these records refers to distances from a given point x to points within X , whether or not x lies in X . As indicated, we use $\text{minDist}(x) + \varepsilon \text{SumDist}(x)$ as the criterion to evaluate a point $x \in Z - X$ to be added to X , by selecting a point that minimizes this criterion. Thus, the ability to quickly update the value of $\text{MinDist}(x)$ and $\text{SumDist}(x)$ without always having to do the full calculations indicated in their preceding definitions saves significant effort. We show next how to establish a complexity bound for the streamlining calculations of $O(|Z|)$ in the average case (the worst case complexity is $O(|Z| \cdot |X|)$, but it is not reached for most iterations).

After adding a point x^* , it is necessary to examine all $x \in Z - \{x^*\}$ and: (i) update $\text{MinDist}(x)$ and initialize $\text{MinDistCount}(x)$ to 1 if $d(x, x^*) < \text{MinDist}(x)$, or (ii) update only $\text{MinDistCount}(x)$ if $d(x, x^*) = \text{MinDist}(x)$, or (iii) do not modify $\text{MinDist}(x)$ or $\text{MinDistCount}(x)$ if $d(x, x^*) > \text{MinDist}(x)$. In addition, $\text{SumDist}(x)$ is always increased by $d(x, x^*)$; the records of x^* do not require any change. All these calculations can always be performed in $O(|Z|)$ using Algorithm 3 below.

Algorithm 3 (Calculation streamlining after adding an element x^*)

```

For Each  $x \in Z - \{x^*\}$ 
     $\text{SumDist}(x) = \text{SumDist}(x) + d(x, x^*)$ 
    If ( $d(x, x^*) = \text{MinDist}(x)$ ) Then
         $\text{MinDistCount}(x) = \text{MinDistCount}(x) + 1$ 
    Else If ( $d(x, x^*) < \text{MinDist}(x)$ )
         $\text{MinDistCount}(x) = 1$ 
         $\text{MinDist}(x) = d(x, x^*)$ 
    End If
End For Each
    
```

The update after dropping point $x^\#$ requires going through all $x \in Z - \{x^\#\}$ and, for each x , one of the following situations may arise:

- (1) do not modify $\text{MinDist}(x)$ or $\text{MinDistCount}(x)$, if $d(x, x^\#) > \text{MinDist}(x)$;
- (2) decrement $\text{MinDistCount}(x)$, if $\text{MinDistCount}(x) > 1$ and $d(x, x^\#) = \text{MinDist}(x)$;
- (3) recalculate $\text{MinDist}(x)$, if $\text{MinDistCount}(x) = 1$ and $d(x, x^\#) = \text{MinDist}(x)$.

Notice that the records of $x^\#$ do not require any modification. Algorithm 4 below presents all update calculations that are executed after dropping $x^\#$.

Algorithm 4 (Update calculations after dropping an element $x^\#$)

```

For Each  $x \in Z - \{x^\#\}$ 
     $\text{SumDist}(x) = \text{SumDist}(x) - d(x, x^\#)$ 
    If ( $d(x, x^\#) = \text{MinDist}(x)$ ) Then
        If ( $\text{MinDistCount}(x) > 1$ ) Then
             $\text{MinDistCount}(x) = \text{MinDistCount}(x) - 1$     (case (2))
        Else
            Re-calculate  $\text{MinDist}(x)$  and  $\text{MinDistCount}(x)$     (case (3))
        Endif
    End If
End For Each
    
```

From all three situations above, case (3) is the most problematic one because it requires recalculating $\text{MinDist}(x)$. Such an update can be performed in linear time for any particular x . In the worst case, all elements $x \in Z - \{x^\#\}$ would require such a recalculation, leading to a quadratic time complexity for the whole Algorithm 4. However, even this theoretically quadratic cost can only reach $O(|Z| \cdot |X|)$ at maximum, less computationally expensive than $O(|Z|^2)$.

Furthermore, in practical terms, the impact of this quadratic worst-case complexity is limited. Indeed, the above $O(|Z| \cdot |X|)$ recalculations are not required at each iteration, because case (3) can not arise systematically. Furthermore, even for the iterations that do require these recalculations, the $O(|Z| \cdot |X|)$ complexity can be reached only if $O(|Z|)$ points

from Z would require updating $\text{MinDist}(x)$ and $\text{MinDistCount}(x)$. Such a situation could only arise if $x^\#$ would be the unique closest point to most of the other points from Z . This condition is quite strong, and consequently, such a high complexity is not often reached in practice. In most of the cases, there is only a bounded number of points $x \in Z$ having $x^\#$ as the unique closest point, and so, we observe that the average complexity of the update operation (over all iterations) is $O(|Z|)$.

Finally, since $k = |X|$ is often significantly smaller than $n = |Z|$ (e.g., in our standard instances, k is either $0.1 \cdot n$ or $0.3 \cdot n$), the implementation should avoid going through all elements of Z when one only needs to iterate through the elements of X . As such, a statement of the form “For each $x \in Z$, if selected[x] then” should be avoided because it has complexity $O(|Z|) > O(|X|)$. For this purpose, it is important to record X both as a simple 0–1 array and as a linked list that can be processed in $O(|X|)$ time with the appropriate data structures.

2.5 Complexity remarks compared with classical approaches

To compute the total iteration complexity, we first observe that Step 1 of Algorithm 2 requires $O(1)$ time. Selecting the best x^* in Step 2 requires going through all elements of $\text{Add } X(k)$ in $O(|Z - X|) < O(|Z|)$ time (for each x , the values $\text{Min}(d(x, y) : y \in X(k - 1))$ and $\sum d(x, y) : y \in X(k - 1)$ are stored in two vectors that are updated by the streamlining routines). Furthermore, the above section showed that the streamlining calculations require $O(|Z|)$ time for most iterations, or $O(|Z| \cdot |X|)$ time in the worst case. As such, the total *worst case iteration complexity* is $O(|Z| \cdot |X|)$, less than $O(|Z|^2)$ in other local search methods. However, compared to previous approaches, the complexity improvement is even more pronounced in the average case, i.e., $O(|Z|)$ compared to $O(|Z|^2)$.

Indeed, most previous local search methods from the literature (see e.g., Erkut 1990 or Ghosh 1996) consider a *quadratic* neighborhood, commonly defined as the set of all potential solutions that can be reached by applying a swap on the current solution, i.e., a neighborhood transition consists of swapping an element $a \in X(k)$ with an element $b \in Z - X(k)$. The evaluation of such a neighborhood requires quadratic time in the worst case *as well as in the average case*. In contrast, most iterations of our algorithm do not require more than $O(|Z|)$ —i.e., only the streamlining routines could require $O(|Z| \cdot |X|)$ and only when the current potential solution verifies certain strong conditions (see Sect. 2.4).

Besides a faster evaluation of a smaller neighborhood, our approach provides easier procedures for ensuring diversification, taking advantage from the tabu mechanism underlying drop moves. (There is obviously no virtue in using a simpler neighborhood if this is not accompanied by procedures that ultimately compensate for potential loss of information that would otherwise be processed at each iteration.) The elementary type of tabu search rule, which defines $\text{Drop } X(k)$ to contain only the “oldest” selected point in $X(k)$, causes each selected point to stay selected for exactly k iterations. A useful feature of this approach is that, after removing point $x^\#$, the algorithm can not immediately add $x^\#$ back because it needs to add a point x^* from $\text{Add } X(k) = Z - X(k)$. If $x^\#$ would be added back in the future, it would be added into a configuration that will have changed in the meantime, and looping would be thus avoided. We also tried using larger sets $\text{Drop } X(k)$, but our experiments showed no significant or conclusive improvement. Indeed, this simple approach appears rather robust relative to the chosen size of $\text{Add } X$ and $\text{Drop } X$.

Furthermore, we also tested more complex techniques such as interleaving Constructive and local search methods, as well as more advanced local search algorithms, e.g., using larger neighborhoods and more complex evaluations of points to add or to drop. While it is

always possible to refine and achieve a certain improvement using such techniques, the goal of this paper is only to present a simple, effective and very practical algorithm: aside from the termination condition (i.e., maxNoGain), one does not even need any parameter tuning.

3 Computational experiments

This section is devoted to computational assessments,¹ as well as to comparisons with the most recent and best performing MMDP approaches: DCGL from Della Croce et al. (2009) and RMGD from Resende et al. (2010). The DCGL approach starts out by sorting the distance values. Each distance value Δ has an associated clique problem that can only be solved if the MMDP optimum is lower than Δ . Based on a powerful clique algorithm, DCGL applies a dichotomic search to rapidly find distance values for which the clique problem can be solved. RMGD proposes a completely different approach based on exploiting existing constructive and local search methods, combined with powerful new techniques based on GRASP (with new local improvement operators), Path-relinking and Evolutionary Path Re-linking.

Following the experimental protocol from the DCGL and RMGD papers, we consider 120 MMDP instances,² belonging to two classes: Geo and Ran. Each of these two classes contains 60 instances that can be further classified into three groups, according to the cardinal of Z : $n = |Z| = 100$, $n = 250$ or $n = 500$. Each group contains 20 instances, half of them with $k = 0.1 \cdot n$, and half with $k = 0.3 \cdot n$.

We will first present the results as summary tables (see Tables 1–5), using the format from the RMGD paper. To be specific, we provide the following three indicators for each group of 20 instances:

Deviation (%) the average deviation from the best MMDP lower bound from the literature (representing the best solution ever obtained by other researchers). For instance, given an instance with a best known lower bound value of 100, a solution of 80 has a deviation of 20%; the average deviation over all instances is reported.

#Best the number of instances for which the best known lower bound is reached.

TimeAvg the average time spent by the algorithm in seconds. The reported CPU times are obtained on a 2.8 GHz Xeon processor using the C++ programming language compiled with the `-O2` optimization option (gcc version 4.1.2 under Linux).

Besides these summaries, we also provide (in Appendix) detailed results, as well as instance-by-instance comparisons with DCGL and RMGD. For supplementary information, we also provide the objective value according to MDP criterion.

The stopping condition is to terminate upon reaching a maximum number of iterations (maxNoGain) with no improvement of the best value of the MMDP^o objective function. We present results with three stopping conditions: $\text{maxNoGain} = 5 \cdot n$, $\text{maxNoGain} = 10000 \cdot n$, and $\text{maxNoGain} = 2000000 \cdot n$. Comparing to other papers from the MMDP literature, these conditions respectively correspond to: very short execution times (< 1 second), moderate execution times (several tens of seconds at maximum) and high execution times (one to several hours). It is worth reiterating that maxNoGain is the only parameter required by our algorithm.

¹The source code of our algorithm and the solutions for the largest instances with $n = 500$ are available at: <http://www.info.univ-angers.fr/pub/hao/mmdp/>.

²Available at <http://www.uv.es/rmarti/paper/mdp.html>.

Table 1 Summary results of the constructive algorithm

Instance group	Deviation	#Best	Avg. Time (sec)
Geo 100	6.41%	0	<0.1
Geo 250	4.10%	0	<0.1
Geo 500	6.11%	0	<0.1
Ran 100	6.17%	0	<0.1
Ran 250	4.56%	0	<0.1
Ran 500	21.40%	0	<0.1
All 120 instances	8.13%	0	<0.1

Table 2 Drop-Add Search summary results for $\text{maxNoGain} = 5 \cdot n$

Instance group	Deviation	#Best	Avg. Time (sec)
Geo 100	1.64%	2	<0.1
Geo 250	1.70%	0	<0.1
Geo 500	2.59%	0	<0.1
Ran 100	3.55%	4	<0.1
Ran 250	2.84%	0	<0.1
Ran 500	11.50%	0	<0.1
All 120 instances	3.97%	6	<0.1

3.1 Results of the constructive algorithm

We first present (Table 1) the summarized results of the constructive algorithm described in Sect. 2.2. This constructive stage stops as soon as k elements are selected.

While this constructive heuristic does not yield solutions of impressive quality compared to those obtained by methods that run for longer durations, the quality is nevertheless noteworthy in view of the amount of time invested in reaching these solutions.

3.2 Results of the Drop-Add search algorithm

The Drop-Add search algorithm is launched from an initial solution provided by the constructive stage. The summarized results, group by group, are presented in Tables 2–4; each table corresponds respectively to one of the three stopping conditions mentioned above.

The first table from these summaries (Table 2) shows that our algorithm can reach quite competitive performances in less than 0.1 seconds. More precisely, given the very low running times associated to the first stopping condition (i.e., $\text{maxNoGain} = 5 \cdot n$), a deviation from the optimum of less than 4% is somewhat surprising, representing a value that might be considered acceptable in many applications.

Table 3 presents the results associated with an intermediate stopping condition ($\text{maxNoGain} = 10000 \cdot n$); our algorithm manages to reach a large proportion (97 over 120) of the best known lower bounds within a moderately short time. For illustration, the Drop-Add search algorithm can solve in 2–3 seconds all benchmark instances with $n = 100$ —i.e., Geo 100 and Ran 100 (the optimality of these bounds is proved in the DCGL paper). In slightly more than 10 seconds, the algorithm also reaches all proved optima of all

Table 3 Drop-Add Search summary results for $\text{maxNoGain} = 10000 \cdot n$

Instance group	Deviation	#Best	Avg. Time (sec)
Geo 100	0.00%	20	2
Geo 250	0.10%	15	13.5
Geo 500	0.31%	8	77.9
Ran 100	0.00%	20	2
Ran 250	0.00%	20	12.3
Ran 500	0.53%	14	58.7
All 120 instances	0.15%	97	27.7

Table 4 Drop-Add Search summary results for $\text{maxNoGain} = 2000000 \cdot n$

Instance group	Deviation	#Best	Avg. Time (sec)
Geo 100	0.00%	20	449.7
Geo 250	0.06%	17	2737
Geo 500	0.11%	16	14093
Ran 100	0.00%	20	454.4
Ran 250	0.00%	20	2499
Ran 500	0.08%	19	12034
All 120 instances	0.04%	112	5378

Ran instances with $n = 250$ with $k = 25$. By allowing longer running times (i.e., using $\text{maxNoGain} = 2000000 \cdot n$), our algorithm reaches 112 best lower bounds out of 120, see Table 4. The average deviation from the best value of MMDP objective function becomes 0.04%, almost negligible.

3.3 Comparison with related heuristics

In order to better evaluate the impact of the proposed ideas, we compare our approach with two related local search methods and a population-based heuristic:

- Swap-LS: A basic local search using a swap-based neighborhood of quadratic size. Essentially, an iteration of this algorithm consists of searching the best swap between a selected and a non-selected element, hence the quadratic complexity of an iteration.
- GhoC+BLS: An algorithm using the GhoC constructive method due to Ghosh (1996), followed by the local search BLS studied in Erkut (1990) and Ghosh (1996). This local search algorithm also utilizes a swap-based quadratic neighborhood.
- Grasp+EvPR: Evolutionary Path-relinking combined with a GRASP algorithm based on a classical constructive method and a significantly improved local search (Resende et al. 2010).

The implementation of Swap-LS is derived by extending the code of our Drop-Add search algorithm. Regarding GhoC+BLS and Grasp+EvPR, we make use of the results provided by the RMGD paper—the detailed results from the RGMD paper are publicly available on-line at heur.uv.es/opticom/mmdp/. However, we needed to recalculate the summaries using the updated bounds provided by the DCGL paper.

Table 5 Comparison between Drop-Add Search (using $\text{maxNoGain} = 10000 \cdot n$, as in Table 3) and three other heuristics using local search

Instance group	Drop-Add Search			Swap-LS			GhoC+BLS			Grasp+EvPR		
	Dev	#Best	Time	Dev	#Best	Time	Dev	#Best	Time	Dev	#Best	Time
	[%]		[s]	[%]		[s]	[%]		[s]	[%]		[s]
Geo 100	0	20	2	0.95	4	2	0.75	10	2	0.09	17	4
Geo 250	0.1	15	13.5	1.29	0	16.05	1.30	0	30	0.47	4	66
Geo 500	0.31	8	77.9	1.4	0	91.6	3.17	0	282	0.88	0	1465
Ran 100	0	20	2	4.42	1	2	1.71	4	1	0.49	15	7
Ran 250	0	20	12.3	3.66	0	12.7	3.26	0	16	1.53	6	271
Ran 500	0.53	14	58.7	21.3	0	66.65	13.52	0	93	11.34	0	6349
All 120 instances	0.15	97	27.7	5.51	5	31.83	3.95	14	71	2.47	42	1360

Table 5 presents the actual comparison between our Drop-Add search algorithm and these three algorithms. The same indicators as in Sect. 3.2 are used: the average deviation from the optimum (Dev [%]), the number of best-known bounds reached (#Best) and the time in seconds (Time [s]). The Drop-Add search algorithm used a very moderate computing time compared to the other three algorithms.

Although the results from Table 5 are not all obtained in the same technical environment (i.e., the first two and the last two algorithms were implemented by different authors on different platforms), they do offer some indications as to the relative performance of the algorithms. Comparing to a quadratic swap-based local search (see Swap-LS, as well as GhoC+BLS), our Drop-Add Search algorithm obtains clearly improved results using a similar or smaller amount of time. Even compared to more complex heuristics based on Evolutionary Path-relinking (hybridized with an enhanced local search), our approach is highly competitive and is able to rapidly reach established bounds in numerous cases (see also the instance-by-instance comparisons in Sect. 3.4).

Notice that in this comparison, the Drop-Add Search is not given an advantage compared to Swap-LS, either in terms of computing time, or in terms of equivalent evaluation counts. Indeed, Table 5 shows that Swap-LS always uses more time than Drop-Add Search (compare columns 7 and 4). This is due to the fact that the stopping condition is more permissive in Swap-LS: while the Drop-Add Search uses $\text{maxNoGain} = 10000 \cdot n$, Swap-LS stops after $\text{maxNoGain} = 15000 \cdot n$ swap evaluations. Since an iteration of Swap-LS consists of searching for the best swap between a selected and an unselected element, such an iteration evaluates a quadratic number of swaps. Consequently, the computational effort of one iteration of our Add-Drop algorithm is roughly equivalent to a single swap evaluation, rather than to a complete (quadratic complexity) Swap-LS iteration. Furthermore, we confirm that the number of swap evaluations always exceeds or is comparable to the number of Drop-Add iterations. Given the difficulty of comparing different methods, the comparisons shown above are provided for indicative purposes and should be interpreted with a word of caution.

3.4 Detailed comparisons and instance-by-instance results

Additional instance-by-instance outcomes are reported in Tables 6 and 7 (Appendix), following the format from the DCGL paper. In both tables we used the intermediate stopping

condition $\text{maxNoGain} = 10000 \cdot n$, as in Tables 3 and 5. The total computing time (tens of seconds, at maximum) can be considered reasonably moderate compared to other methods from the literature. Indeed, our algorithm required an average time of 31.17 seconds for the Geo instances and an average time of 24.35 seconds for the Ran instances. The best methods among those tested by the RMGD paper required hundreds of seconds; the DCGL algorithm required average times of tens of seconds as well.

Table 6 shows that the Drop-Add search algorithm achieves a level of performance on a par with that of the DCGL approach—although DCGL utilizes as an internal component a more complex and refined max-clique algorithm presented in Grosso, Locatelli and Pullan (2008). Furthermore, Table 7 presents an instance-by-instance comparison with the best bounds reported by the RMGD paper (also, individual results of two algorithms from RMGD are summarized Table 5). Besides the fact that our algorithm is very fast,³ it always reaches the best RMGD bounds and even improves upon these best bounds in more than 60 instances.

4 Conclusions

We have presented a simple approach for MMDP that nevertheless proves surprisingly effective. For example, the proposed Drop-Add Simple Tabu Search finds within several seconds all optimally-proved solutions for all standard benchmark instances with $n = 100$, as well as for certain instances with $n = 250$. The algorithm is competitive with more refined approaches from the literature, and even outperforms several methods tested in Resende et al. (2010), e.g., methods that refer to Simulated Annealing, Tabu Search (based on a different foundation, using classical swap-based quadratic neighborhoods), GRASP with path-relinking, GRASP with evolutionary path-relinking, etc. By allowing reasonably larger amounts of time, our approach reached almost all best-known lower bounds—reported by the algorithm of Della Croce et al. (2009). We also report a solution better than the previous best known solution to the benchmark instance “Ran 500 3”, establishing a new MMDP lower bound of 56 (the previous best-known bound was 55).

As detailed in Sect. 2.5, the effectiveness of the new approach, which utilizes an extremely simple type of tabu search rule, derives from a low iteration complexity and from strong diversification qualities of the search process. Indeed, apart from the streamlining calculations that rarely require more than linear time, the complexity of an iteration is $O(|Z|)$ —computationally less expensive than $O(|Z|^2)$ in other local search-based algorithms. Besides such speed benefits, important diversification properties arise from the fact that the simple tabu rule causes each selected element to be systematically dropped after a fixed period of time.

Acknowledgements We are grateful to the reviewers of the paper for their helpful comments and suggestions. This work was partially supported by the Region of “Pays de la Loire” (France) within the Radapop (2009-2012) and LigeRO (2010-2013) Projects.

³We do not claim that such speed comparisons are absolute. Recall our reported CPU times are obtained on a 2.8 GHz Xeon processor using the C++ programming language compiled with the -O2 optimization flag (gcc version 4.1.2 under Linux). RMGD and DCGL used Pentium IV processors running at 3–3.2 GHz.

Appendix

Table 6 Objective function values reached by our local search algorithm (i.e. using $\text{maxNoGain} = 10.000 \cdot n$, see Sect. 3.2), compared to those reported by the DCGL algorithm proposed by Della Croce et al. (2009). Our values of the MMDP objective function (Column 4) are marked in bold (in case of equality) or in bold and underlined (if strictly better bounds are reached). Seven columns are reported: (1) the instance; (2) n , the number of vertices; (3) k , the number of elements to be selected; (4) our value of the MMDP objective function; (5) our value of the MDP objective function; (6) the time in seconds; (7) the best bound from the DCGL paper

Instance	n	k	MMDP	MDP	Time [s]	DCGL bound
Geo 100 1	100	10	89.37	5216.44	2	89.37
Geo 100 2	100	10	96.66	5333.46	2	96.66
Geo 100 3	100	10	76.45	4792.48	2	76.45
Geo 100 4	100	10	103.01	5756.19	2	103.01
Geo 100 5	100	10	119.72	6449.07	2	119.72
Geo 100 6	100	10	36.52	3179.37	2	36.52
Geo 100 7	100	10	189.21	9481.21	2	189.21
Geo 100 8	100	10	110.02	6058.5	2	110.02
Geo 100 9	100	10	141.24	7419.54	2	141.24
Geo 100 10	100	10	163.68	8479.88	2	163.68
Geo 100 11	100	30	102.19	62515.3	2	102.19
Geo 100 12	100	30	117.48	68037.1	2	117.48
Geo 100 13	100	30	29.83	31830.6	2	29.83
Geo 100 14	100	30	54.08	43874.4	2	54.08
Geo 100 15	100	30	144.48	81089.8	2	144.48
Geo 100 16	100	30	122.38	72309.5	2	122.38
Geo 100 17	100	30	130.35	75325.3	2	130.35
Geo 100 18	100	30	109.87	66688.4	2	109.87
Geo 100 19	100	30	152.21	83503.3	2	152.21
Geo 100 20	100	30	<u>140.605</u>^a	78442.9	2	140.6
Geo 250 1	250	25	171.01	61780.7	12	171.01
Geo 250 2	250	25	20	17769.9	13	20.03
Geo 250 3	250	25	137.75	51211.5	12	137.75
Geo 250 4	250	25	171.96	61719	12	171.96
Geo 250 5	250	25	148.76	55424.7	12	148.76
Geo 250 6	250	25	100.38	40922.5	12	100.38
Geo 250 7	250	25	175.71	63348.9	12	175.71
Geo 250 8	250	25	164.51	58745.7	12	164.51
Geo 250 9	250	25	179.18	64817.5	12	179.18
Geo 250 10	250	25	102.23	42373.5	12	102.23
Geo 250 11	250	75	31.37	239167	13	31.72
Geo 250 12	250	75	93.91	407763	16	93.91
Geo 250 13	250	75	145.13	539513	14	145.22
Geo 250 14	250	75	70.32	344550	25	70.7
Geo 250 15	250	75	142.54	537908	17	142.54

Table 6 (Continued)

Instance	n	k	MMDP	MDP	Time [s]	DCGL bound
Geo 250 16	250	75	108.05	453951	13	108.05
Geo 250 17	250	75	124.42	486532	13	124.63
Geo 250 18	250	75	148.76	553873	13	148.76
Geo 250 19	250	75	134.74	517918	13	134.74
Geo 250 20	250	75	147.83	552305	13	147.83
Geo 500 1	500	50	124.79	206631	65	124.79
Geo 500 2	500	50	13.56	70268.6	65	13.79
Geo 500 3	500	50	164.91	252796	56	165.04
Geo 500 4	500	50	132.5	215553	56	132.5
Geo 500 5	500	50	28.18	93069.6	83	28.55
Geo 500 6	500	50	28.18	92420.5	123	28.6
Geo 500 7	500	50	132.51	215393	69	132.51
Geo 500 8	500	50	113.586	193055	81	113.6
Geo 500 9	500	50	168.96	257865	63	168.96
Geo 500 10	500	50	159.98	248663	69	159.98
Geo 500 11	500	150	93.66	1.71314e+06	118	93.97
Geo 500 12	500	150	71.26	1.46952e+06	72	71.46
Geo 500 13	500	150	134.37	2.1535e+06	80	134.47
Geo 500 14	500	150	111.43	1.90051e+06	69	111.63
Geo 500 15	500	150	35.97	1.0717e+06	71	36.18
Geo 500 16	500	150	132.58	2.12167e+06	112	132.58
Geo 500 17	500	150	129.3	2.09709e+06	74	129.49
Geo 500 18	500	150	72.49	1.48543e+06	66	72.65
Geo 500 19	500	150	123.99	2.03106e+06	74	123.99
Geo 500 20	500	150	123.43	2.02141e+06	93	123.43
Ran 100 1	100	10	73	3957	2	73
Ran 100 2	100	10	75	3910	2	75
Ran 100 3	100	10	74	3975	2	74
Ran 100 4	100	10	74	3962	2	74
Ran 100 5	100	10	74	3858	2	74
Ran 100 6	100	10	74	4000	2	74
Ran 100 7	100	10	75	3914	2	75
Ran 100 8	100	10	74	3969	2	74
Ran 100 9	100	10	74	3969	2	74
Ran 100 10	100	10	75	3968	2	75
Ran 100 11	100	30	54	34740	2	54
Ran 100 12	100	30	55	34625	2	55
Ran 100 13	100	30	55	34319	2	55
Ran 100 14	100	30	55	34455	2	55
Ran 100 15	100	30	55	34799	2	55
Ran 100 16	100	30	55	34064	2	55
Ran 100 17	100	30	55	34218	2	55
Ran 100 18	100	30	55	34089	2	55
Ran 100 19	100	30	55	34654	2	55

Table 6 (Continued)

Instance	n	k	MMDP	MDP	Time [s]	DCGL bound
Ran 100 20	100	30	55	34326	2	55
Ran 250 1	250	25	61	24486	12	61
Ran 250 2	250	25	61	24464	13	61
Ran 250 3	250	25	61	24686	12	61
Ran 250 4	250	25	61	24108	13	61
Ran 250 5	250	25	61	24528	12	61
Ran 250 6	250	25	61	24553	12	61
Ran 250 7	250	25	61	24246	12	61
Ran 250 8	250	25	61	24337	12	61
Ran 250 9	250	25	61	24087	14	61
Ran 250 10	250	25	61	24295	12	61
Ran 250 11	250	75	52	213848	12	52
Ran 250 12	250	75	52	213534	13	52
Ran 250 13	250	75	52	214402	12	52
Ran 250 14	250	75	52	213365	12	52
Ran 250 15	250	75	52	213545	13	52
Ran 250 16	250	75	52	214506	12	52
Ran 250 17	250	75	52	213992	12	52
Ran 250 18	250	75	52	214225	12	52
Ran 250 19	250	75	52	213496	13	52
Ran 250 20	250	75	52	214591	12	52
Ran 500 1	500	50	55	97889	54	55
Ran 500 2	500	50	55	97992	53	56
Ran 500 3	500	50	55	97865	53	55
Ran 500 4	500	50	55	97869	53	56
Ran 500 5	500	50	55	97890	53	56
Ran 500 6	500	50	55	97912	54	55
Ran 500 7	500	50	56	96797	64	56
Ran 500 8	500	50	55	97543	53	56
Ran 500 9	500	50	55	98196	53	56
Ran 500 10	500	50	55	98049	53	56
Ran 500 11	500	150	5	1.16892e+06	63	5
Ran 500 12	500	150	5	1.17127e+06	65	5
Ran 500 13	500	150	5	1.17047e+06	63	5
Ran 500 14	500	150	5	1.17303e+06	63	5
Ran 500 15	500	150	5	1.17401e+06	63	5
Ran 500 16	500	150	5	1.17475e+06	62	5
Ran 500 17	500	150	5	1.1768e+06	63	5
Ran 500 18	500	150	5	1.17857e+06	63	5
Ran 500 19	500	150	5	1.17798e+06	63	5
Ran 500 20	500	150	5	1.1832e+06	63	5

^aIn this case, the difference from the previous best-known bound is less than 0.005. We consider that such differences are rather due to the precision of the floating-point calculations (a recurrent issue in such a context)

Table 7 Objective function values reached by our local search algorithm (i.e. $\max\text{NoGain} = 10.000 \cdot n$, see Sect. 3.2), compared to the best bounds reported by the RMGD paper Resende et al. (2010). These RMGD bounds were always reached, and so, all values from Column 4 are marked in bold. Our approach reaches better bounds for more than 60 instances—see that our objective function value is underlined when it is larger than the one from Column 7. Our MMDP bounds are the same as those from Table 6 and those summarized in Table 3

Instance	n	k	MMDP	MDP	Time [s]	RMGD bound
Geo 100 1	100	10	89.37	5216.44	2	89.37
Geo 100 2	100	10	96.66	5333.46	2	96.66
Geo 100 3	100	10	76.45	4792.48	2	76.45
Geo 100 4	100	10	103.01	5756.19	2	103.01
Geo 100 5	100	10	119.72	6449.07	2	119.72
Geo 100 6	100	10	36.52	3179.37	2	36.52
Geo 100 7	100	10	189.21	9481.21	2	189.21
Geo 100 8	100	10	110.02	6058.5	2	110.02
Geo 100 9	100	10	141.24	7419.54	2	141.24
Geo 100 10	100	10	163.68	8479.88	2	163.68
Geo 100 11	100	30	102.19	62515.3	2	102.19
Geo 100 12	100	30	117.48	68037.1	2	117.48
Geo 100 13	100	30	29.83	31830.6	2	29.83
Geo 100 14	100	30	54.08	43874.4	2	54.08
Geo 100 15	100	30	144.48	81089.8	2	144.48
Geo 100 16	100	30	122.38	72309.5	2	122.38
Geo 100 17	100	30	130.35	75325.3	2	130.35
Geo 100 18	100	30	109.87	66688.4	2	109.87
Geo 100 19	100	30	152.21	83503.3	2	152.21
Geo 100 20	100	30	<u>140.605</u>	78442.9	2	140.6
Geo 250 1	250	25	171.01	61780.7	12	171.01
Geo 250 2	250	25	<u>20.0023</u>	17769.9	13	19.7
Geo 250 3	250	25	137.75	51211.5	12	137.75
Geo 250 4	250	25	<u>171.961</u>	61719	12	171.21
Geo 250 5	250	25	<u>148.76</u>	55424.7	12	148.72
Geo 250 6	250	25	<u>100.377</u>	40922.5	12	99.8
Geo 250 7	250	25	175.71	63348.9	12	175.71
Geo 250 8	250	25	<u>164.513</u>	58745.7	12	164.03
Geo 250 9	250	25	<u>179.178</u>	64817.5	12	179.16
Geo 250 10	250	25	<u>102.227</u>	42373.5	12	102.19
Geo 250 11	250	75	31.37	239167	13	31.37
Geo 250 12	250	75	<u>93.9123</u>	407763	16	93.84
Geo 250 13	250	75	<u>145.133</u>	539513	14	145.06
Geo 250 14	250	75	<u>70.3202</u>	344550	25	70.12
Geo 250 15	250	75	<u>142.544</u>	537908	17	142.5
Geo 250 16	250	75	108.05	453951	13	108.05
Geo 250 17	250	75	<u>124.422</u>	486532	13	124.25

Table 7 (Continued)

Instance	n	k	MMDP	MDP	Time [s]	RMGD bound
Geo 250 18	250	75	<u>148.762</u>	553873	13	148.63
Geo 250 19	250	75	<u>134.736</u>	517918	13	133.87
Geo 250 20	250	75	147.83	552305	13	147.83
Geo 500 1	500	50	<u>124.789</u>	206631	65	123.74
Geo 500 2	500	50	<u>13.5627</u>	70268.6	65	13.4
Geo 500 3	500	50	<u>164.909</u>	252796	56	164.13
Geo 500 4	500	50	<u>132.501</u>	215553	56	131.62
Geo 500 5	500	50	<u>28.1795</u>	93069.6	83	28.07
Geo 500 6	500	50	<u>28.1815</u>	92420.5	123	27.8
Geo 500 7	500	50	<u>132.514</u>	215393	69	131.34
Geo 500 8	500	50	<u>113.586</u>	193055	81	112.68
Geo 500 9	500	50	<u>168.959</u>	257865	63	168.24
Geo 500 10	500	50	<u>159.976</u>	248663	69	159.68
Geo 500 11	500	150	<u>93.6574</u>	1.71314e+06	118	93.49
Geo 500 12	500	150	<u>71.2575</u>	1.46952e+06	72	71.12
Geo 500 13	500	150	<u>134.37</u>	2.1535e+06	80	133.99
Geo 500 14	500	150	<u>111.43</u>	1.90051e+06	69	111.04
Geo 500 15	500	150	<u>35.9653</u>	1.0717e+06	71	35.71
Geo 500 16	500	150	<u>132.577</u>	2.12167e+06	112	132.43
Geo 500 17	500	150	<u>129.3</u>	2.09709e+06	74	129.04
Geo 500 18	500	150	<u>72.4858</u>	1.48543e+06	66	71.85
Geo 500 19	500	150	<u>123.986</u>	2.03106e+06	74	123.95
Geo 500 20	500	150	<u>123.429</u>	2.02141e+06	93	123.14
Ran 100 1	100	10	73	3957	2	73
Ran 100 2	100	10	75	3910	2	75
Ran 100 3	100	10	74	3975	2	74
Ran 100 4	100	10	74	3962	2	74
Ran 100 5	100	10	74	3858	2	74
Ran 100 6	100	10	74	4000	2	74
Ran 100 7	100	10	75	3914	2	75
Ran 100 8	100	10	74	3969	2	74
Ran 100 9	100	10	74	3969	2	74
Ran 100 10	100	10	75	3968	2	75
Ran 100 11	100	30	54	34740	2	54
Ran 100 12	100	30	55	34625	2	55
Ran 100 13	100	30	55	34319	2	55
Ran 100 14	100	30	55	34455	2	55
Ran 100 15	100	30	55	34799	2	55
Ran 100 16	100	30	55	34064	2	55
Ran 100 17	100	30	55	34218	2	55
Ran 100 18	100	30	55	34089	2	55

Table 7 (Continued)

Instance	n	k	MMDP	MDP	Time [s]	RMGD bound
Ran 100 19	100	30	55	34654	2	55
Ran 100 20	100	30	55	34326	2	55
Ran 250 1	250	25	61	24486	12	59
Ran 250 2	250	25	61	24464	13	60
Ran 250 3	250	25	61	24686	12	60
Ran 250 4	250	25	61	24108	13	59
Ran 250 5	250	25	61	24528	12	60
Ran 250 6	250	25	61	24553	12	60
Ran 250 7	250	25	61	24246	12	60
Ran 250 8	250	25	61	24337	12	60
Ran 250 9	250	25	61	24087	14	60
Ran 250 10	250	25	61	24295	12	60
Ran 250 11	250	75	52	213848	12	52
Ran 250 12	250	75	52	213534	13	51
Ran 250 13	250	75	52	214402	12	52
Ran 250 14	250	75	52	213365	12	52
Ran 250 15	250	75	52	213545	13	51
Ran 250 16	250	75	52	214506	12	52
Ran 250 17	250	75	52	213992	12	52
Ran 250 18	250	75	52	214225	12	52
Ran 250 19	250	75	52	213496	13	52
Ran 250 20	250	75	52	214591	12	51
Ran 500 1	500	50	55	97889	54	54
Ran 500 2	500	50	55	97992	53	55
Ran 500 3	500	50	55	97865	53	55
Ran 500 4	500	50	55	97869	53	55
Ran 500 5	500	50	55	97890	53	54
Ran 500 6	500	50	55	97912	54	54
Ran 500 7	500	50	56	96797	64	54
Ran 500 8	500	50	55	97543	53	55
Ran 500 9	500	50	55	98196	53	55
Ran 500 10	500	50	55	98049	53	54
Ran 500 11	500	150	5	1.16892e+06	63	4
Ran 500 12	500	150	5	1.17127e+06	65	4
Ran 500 13	500	150	5	1.17047e+06	63	4
Ran 500 14	500	150	5	1.17303e+06	63	4
Ran 500 15	500	150	5	1.17401e+06	63	4
Ran 500 16	500	150	5	1.17475e+06	62	4
Ran 500 17	500	150	5	1.1768e+06	63	4
Ran 500 18	500	150	5	1.17857e+06	63	4
Ran 500 19	500	150	5	1.17798e+06	63	4
Ran 500 20	500	150	5	1.1832e+06	63	4

References

- Andrade, M., Andrade, P., Martins, S., & Plastino, A. (2005). *Lecture notes in computer science: Vol. 3503. GRASP with path-relinking for the maximum diversity problem* (pp. 558–569).
- April, J., Glover, F., Kelly, J. P., & Laguna, M. (2003). Simulation-based optimization: practical introduction to simulation optimization. In *Proceedings of the 35th conference on Winter simulation: driving innovation* (pp. 71–78).
- Aringhieri, R., & Cordone, R. (2011). Comparing local search metaheuristics for the maximum diversity problem. *The Journal of the Operational Research Society*, 62(2), 266–280.
- Aringhieri, R., Cordone, R., & Melzani, Y. (2008). Tabu search versus GRASP for the maximum diversity problem. *4OR: A Quarterly Journal of Operations Research*, 6(1), 45–60.
- Della Croce, F., Grosso, A., & Locatelli, M. (2009). A heuristic approach for the max-min diversity problem based on max-clique. *Computers and Operations Research*, 36(8), 2429–2433.
- Duarte, A., & Marti, R. (2007). Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 178, 71–84.
- Erkut, E. (1990). The discrete dispersion problem. *European Journal of Operational Research*, 46, 48–60.
- Gallego, M., Duarte, A., Laguna, M., & Marti, R. (2009). Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, 44(3), 411–426.
- Ghosh, J. B. (1996). Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19, 175–181.
- Glover, F., & Laguna, M. (1997). *Tabu search*. Dordrecht: Kluwer Academic.
- Glover, F., Kuo, C., & Dhir, K. (1998). Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1), 109–132.
- Grosso, A., Locatelli, M., & Pullan, W. (2008). Randomness, plateau search, penalties, restart rules: simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6), 587–612.
- Kincaid, R. (1992). Good solutions to discrete noxious location problems via metaheuristics. *Annals of Operation Research*, 40, 265–281.
- Kuo, C., Glover, F., & Dhir, K. (1993). Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6), 1171–1185.
- Östergård, P. R. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1–3), 197–207.
- Palubeckis, G. (2007). Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 189(1), 371–383.
- Resende, M., Marti, R., Gallego, M., & Duarte, A. (2010). GRASP and path relinking for the max-min diversity problem. *Computers and Operations Research*, 37(3), 498–408.
- Santos, L., Martins, S., & Plastino, A. (2008). Applications of the DM-GRASP heuristic: a survey. *International Transactions in Operational Research*, 15(4), 387–416.
- Silva, G. C., Ochi, L. S., & Martins, S. L. (2004). *Lecture notes in computer science: Vol. 3059. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem* (pp. 498–512).
- Wang, J., Zhou, Y., Yin, J., & Zhang, Y. (2009). Competitive hopfield network combined with estimation of distribution for maximum diversity problems. *IEEE Transactions on Systems, Man and Cybernetics. Part B. Cybernetics*, 39(4), 1048–1066.