**Construction and improvement algorithms for dispersion problems**


# Construction and improvement algorithms for dispersion problems

Roberto Aringhieri, Roberto Cordone, Andrea Grosso

PII: S0377-2217(14)00797-8
DOI: 10.1016/j.ejor.2014.09.058
Reference: EOR 12545

To appear in: *European Journal of Operational Research*

Received date: 3 August 2013
Revised date: 22 September 2014
Accepted date: 24 September 2014

Please cite this article as: Roberto Aringhieri, Roberto Cordone, Andrea Grosso, Construction and improvement algorithms for dispersion problems, *European Journal of Operational Research* (2014), doi: 10.1016/j.ejor.2014.09.058

# Construction and improvement algorithms for dispersion problems

Roberto Aringhieri[a], Roberto Cordone[b,*], Andrea Grosso[a]

[a] *Università degli Studi di Torino*
*Dipartimento di Informatica*
*Corso Svizzera, 185 - 10149 Torino, Italy*
[b] *Università degli Studi di Milano*
*Dipartimento di Informatica*
*Via Comelico, 39 - 20135 Milano, Italy*

**Abstract**

Given a set $N$, a pairwise distance function $d$ and an integer number $m$, the *Dispersion Problems* (*DP*s) require to extract from $N$ a subset $M$ of cardinality $m$, so as to optimize a suitable function of the distances between the elements in $M$. Different functions give rise to a whole family of Combinatorial Optimization problems. In particular, the *max-sum DP* and the *max-min DP* have received strong attention in the literature. Other problems (e. g., the *max-minsum DP* and the *min-diffsum DP*) have been recently proposed with the aim to model the optimization of equity requirements, as opposed to that of more classical efficiency requirements. Building on the main ideas which underly some state-of-the-art methods for the *max-sum DP* and the *max-min DP*, this work proposes some constructive procedures and a Tabu Search algorithm for the new problems. In particular, we investigate the extension to the new context of key features such as initialization, tenure management and diversification mechanisms. The computational experiments show that the algorithms applying these ideas perform effectively on the publicly available benchmarks, but also that there are some interesting differences with respect to the *DP*s more studied in the literature. As a result of this investigation, we also provide optimal results and bounds as a

*Corresponding author
*Email addresses:* roberto.aringhieri@unito.it (Roberto Aringhieri), roberto.cordone@unimi.it (Roberto Cordone), andrea.grosso@unito.it (Andrea Grosso)

useful reference for further studies.

*Keywords:* Combinatorial Optimization, Dispersion problems, Binary quadratic programming, Tabu Search

## 1. Introduction

Let $N$ be a set of $n$ elements, $m$ a positive integer number smaller than $n$ and $d : N \times N \to \mathbb{R}$ a distance function on the elements of $N$, such that $d_{ii} = 0$ for all $i \in N$, $d_{ij} \geq 0$ and $d_{ij} = d_{ji}$ for all $i, j \in N$. The literature denotes by *Dispersion Problems* (*DP*s) a family of problems which require to extract from $N$ a subset $M$ of cardinality $m$, so as to optimize a suitable function of the distances between the extracted elements (Erkut, 1990; Prokopyev et al., 2009). The natural mathematical programming formulations for these problems associate a binary variable $x_i$ to each element $i \in N$ and set $x_i = 1$ if $i$ belongs to $M$, $x_i = 0$ otherwise:

$$\max z = f_d(x) \tag{1a}$$

$$\text{s.t.} \sum_{i \in N} x_i = m \tag{1b}$$

$$x_i \in \{0, 1\} \qquad i \in N \tag{1c}$$

where notation $f_d(x)$ means that the objective is a composite function of vector $x$ *through* the distance function $d$, and specifically it depends only on the distances $d_{ij}$ between pairs of elements $(i, j)$ such that $x_i = x_j = 1$.

A whole family of problems can be derived from (1) by specifying in different ways the expression of $f_d(\cdot)$. All of them share the same set of feasible solutions, but the properties and behaviour of their objective functions can be strongly different. In particular, the classical application of *DP*s has been the maximization of some dispersion index used as a measure of operational efficiency. This may refer to the location of facilities (Kuby, 1987; Erkut and Neuman, 1989), but also to the protection of biological diversity, the formulation of admission policies, the formation of committees, the composition of medical crews (Adil and Ghosh, 2005; Glover et al., 1998; Kuo et al., 1993; Weitz and Lakshminarayanan, 1998; Aringhieri, 2009) and, more theoretically, the identification of densest subgraphs (Brimberg et al., 2009). For example, the *max-sum DP*, more commonly known as *Maximum Diversity Problem* (*MDP*) aims to maximize the sum of the pairwise distances between

all selected elements (Kuo et al., 1993):

$$f_d(x) = \frac{1}{2} \sum_{i,j \in N} d_{ij} x_i x_j \tag{2}$$

whereas the *max-min DP* aims to maximize the distance between the two closest elements (Erkut, 1990):

$$f_d(x) = \min_{i,j \in N: x_i = x_j = 1} d_{ij} x_i x_j \tag{3}$$

In contrast to the classical line of research, Erkut and Neuman (1991) and Prokopyev et al. (2009) introduced alternative definitions of $f_d(\cdot)$ to express equity requirements, referring in particular to the idea of fairness among candidate sites for urban public facilities. This alternative approach is focused on an intermediate *aggregate dispersion* measure

$$D_i(x) = \sum_{j \in N} d_{ij} x_j \qquad i \in N \tag{4}$$

that is the sum of the distances between each single element and the selected ones, and can be equivalently expressed as $\sum_{j \in M} d_{ij}$. Both papers investigate the optimization of the *max-minsum DP*, which aims to guarantee that each selected element is as distant as possible from the other ones, setting:

$$f_d(x) = \min_{i \in N: x_i = 1} D_i(x) \tag{5}$$

which is the minimum aggregate dispersion for the selected elements, and which should be maximized. Prokopyev et al. (2009) also consider the *min-diffsum DP*, which aims to guarantee that each selected element has approximately the same total distance from the other ones, setting

$$f_d(x) = \max_{i \in N: x_i = 1} D_i(x) - \min_{j \in N: x_j = 1} D_j(x) \tag{6}$$

that is the maximum difference between the aggregate dispersions of the selected elements. Notice that function $f_d(\cdot)$ should be minimized here, instead of maximized.

This work proposes heuristic algorithms for the *max-minsum DP* and the *min-diffsum DP*, inspired by the state-of-the-art methods for the *max-sum DP* and *max-min DP*.

Our first aim is to investigate whether the main ideas which guarantee a strong performance on efficiency-concerned *DP*s maintain their effectiveness when applied to equity-concerned *DP*s. On one hand, the identical structure of the feasible set for these two subfamilies of *DP*s suggests that it might be the case. On the other hand, the completely different structure of the objective function, and consequently of the so called *landscape* of the problem (Stadler, 1992), poses reasonable doubts on this assumption. See also Resende et al. (2010) for a discussion on the weak correlation between the optimal solutions of the *max-sum DP* and the *max-min DP*. From this perspective, we investigate the impact of some constructive heuristics based on the idea of determining a more favourable initial solution and compare them with the use of a random restart procedure as in Aringhieri and Cordone (2011).

The second aim of this work is to provide best known results for publicly available benchmark instances of equity-concerned *DP*s, thus stimulating further research on the topic, as done for the *max-sum DP* in Martì et al. (2011). Since the instances considered in Prokopyev et al. (2009) are not publicly available and their size ($50 - 100$ elements) is currently too small for a significant algorithmic comparison, we adopted the benchmark instances of the *Optsicom* web site (`http://www.optsicom.es/mdp`). These were originally proposed for the *max-sum DP*, but can be directly employed for all *DP*s. Specifically, we consider instances up to 500 elements. For some of them, we also provide optimal results, or at least bounds, obtained with a general purpose Mixed Integer Linear Programming (*MILP*) solver.

The paper is organized as follows. Section 2 surveys the relevant literature. Section 3 presents the algorithms here proposed to solve the *max-minsum DP* and the *min-diffsum DP*, discussing in detail the basic ideas inspired by the literature on efficiency-concerned *DP*s. Section 4 reports and discusses the computational results. The appendix reports the best known results on all the tested instances.

## 2. A survey on dispersion problems

Most of the literature on *DP*s concerns the *max-sum DP* and the *max-min DP*. Briefly summarizing, the exact methods for the *max-sum DP* can solve instances up to $100 - 150$ elements (Erkut, 1990; Pisinger, 2006; Aringhieri et al., 2009; Martì et al., 2010), whereas larger instances require heuristic approaches. Most of these approaches are local search metaheuristics based on

the simple exchange of elements in and out of the current solution. In particular, the hybrid metaheuristic proposed by Wu and Hao (2013) provides the best known results for a large set of benchmark instances, whose size goes up to 5 000 elements. Other approaches exhibiting remarkable performances are Variable Neighbourhood Search (Brimberg et al., 2009; Aringhieri and Cordone, 2011), Iterated Tabu Search (Palubeckis, 2007), Learnable Tabu Search (Wang et al., 2012), basic Tabu Search (Duarte and Martì, 2007; Aringhieri et al., 2008), Scatter Search (Gallego et al., 2009; Aringhieri and Cordone, 2011) and *GRASP* (Duarte and Martì, 2007; Silva et al., 2007). According to our experience on this problem, the key to impressively good performances and fast execution times is the use of strong, though not necessarily sophisticated, diversification mechanisms (Aringhieri and Cordone, 2011).

The *max-min DP*, on the other hand, suffers from a very flat landscape of the objective function (several different solutions have exactly the same value). This poses a severe challenge on local search metaheuristics, as discussed in Resende et al. (2010), where different heuristics are extensively compared, among which a *GRASP* with evolutionary Path Relinking exhibits the best performance. To partly overcome the issue of the flat landscape, this work minimizes also a secondary objective function, that is the number of pairs $(i, j)$ in the solution such that $d_{ij}$ is minimum. Della Croce et al. (2009) reformulate the *max-min DP* as a dichotomic search on a sequence of instances of the *Maximum Clique Problem* (*MCP*), which are solved with the powerful Iterated Local Search (*ILS*) heuristic proposed in Grosso et al. (2008). This approach allows to prove the optimality of the solution for several instances up to $n = 250$ elements, provided that the clique subproblem is solved with an exact algorithm. In the end, the Tabu Search algorithm described in Porumbel et al. (2011) applies separate add and drop operations to reduce the complexity of each iteration from quadratic to linear, and it adopts an extremely simple tabu rule: the drop operation always removes the oldest selected element. In this way, each element remains in the solution for exactly $m$ iterations. The algorithm also exploits the sum of all pairwise distances between the elements of the solution as an auxiliary objective function to perturb the flat landscape of the problem.

Switching to equity-concerned models, the *max-minsum DP* has been introduced by Erkut and Neuman (1991) and the *min-diffsum DP* by Prokopyev et al. (2009), who provide *MILP* formulations and discuss the computational complexity of both problems, and of other related ones. They also apply a

general-purpose solver on instances from 30 to 100 elements and a GRASP metaheuristic on the instances with 50 and 100 elements. This algorithm generates a starting solution adding one element at a time, chosen randomly from a restricted candidate list of random length, which includes the elements yielding the best partial solutions. Then, the starting solution is improved with a sort of first-improvement local search on a restricted neighbourhood. This attempts a random exchange between one element in the solution and one out of it: if the objective improves, the new solution is accepted and the search restarts from it. If it is rejected for a specified number of times, the improvement phase gives place to a new constructive phase. The whole algorithm terminates after a given number of constructive and improvement phases.

Some works on equity-concerned models (Prokopyev et al., 2009) allow the distance function $d$ to assume also negative values. Notice that, due to the cardinality constraint, if all distances between two distinct elements are increased by a uniform amount $\delta > 0$, the value of each feasible solution correspondingly increases by a fixed amount depending on $\delta$ and on the cardinality $m$:

- for the *max-sum DP*, objective function (2) increases by $m\,(m-1)\,\delta/2$;
- for the *max-min DP*, objective function (3) increases by $\delta$;
- for the *max-minsum DP*, objective function (5) increases by $(m-1)\,\delta$, because all aggregate dispersions (4) increase by $(m-1)\,\delta$;
- for the *min-diffsum DP*, objective function (6) remains unmodified, because all aggregate dispersions (4) increase by $(m-1)\,\delta$.

This implies the following simplifying remark.

**Remark 1.** *The assumption that the distance function d is nonnegative does not introduce any loss of generality.*

A problem related to the ones here considered is the *Equitable Dispersion Problem*, or *max-mean DP*, which maximizes the average distance between the elements of the solution. This problem, contrary to the ones here considered, does not impose a cardinality constraint. The assumption that the distance function can assume both positive and negative values, then, becomes crucial. Martí and Sandoya (2012) propose a GRASP algorithm with Path Relinking for the *max-mean DP*, applying it to instances up to 500 elements, drawn from the *Optsicom* web site.

## 3. Algorithms for equity-concerned dispersion problems

This section describes a number of two-phase algorithms for the the *max-minsum DP* and the *min-diffsum DP* inspired by the authors' work on *max-sum* and the *max-min DP*. The constructive phase exploits the relation between these problems and the *MCP*, computing a sequence of candidate solutions as maximum cliques on a progressively sparsified auxiliary graph (Section 3.1). The improvement phase is a Tabu Search procedure based on the exchange of an element belonging to the current solution with an element out of it (Section 3.2). The two phases repeat iteratively; at each iteration, a simple diversification mechanism rules the constructive phase so as to differentiate the solution built from the one returned by the previous improvement phase (Section 3.3).

### *3.1. Constructive phase*

As discussed in Prokopyev et al. (2009), several *DP*s are strongly $\mathcal{NP}$-hard. The reduction associates the vertices of a *MCP* instance to the elements of a *DP* instance, and the edges to suitable pairs of elements, such as those with a large distance.

**Algorithm 1:** *BuildDP*$(N, d, m, DP)$

```
begin
    E := N × N; M* := ∅; f* := 0;
    repeat
        (N, E) := Sparsify(N, E, d, DP);
        M := FindClique(N, E, m);
        if M ≠ ∅ then
            f := EvaluateSolution(M, DP);
            if Better(f, f*, DP) then M* := M; f* := f;
    until M = ∅;
Output: M*;
```

Procedure *BuildDP* (see Algorithm 1) heuristically exploits this concept. It receives the element set $N$, the distance function $d$, the desired cardinality $m$ and the indication of the problem to solve (*max-minsum DP* or *min-diffsum DP*). After building an auxiliary complete graph $G = (N, E)$ and an empty best known solution $M^*$ of value $f^* = 0$, it applies procedure *Sparsify* to trim $G$ removing its less promising components, as explained

later in detail. Procedure *FindClique* applies the *ILS* heuristic of Grosso et al. (2008) to find a clique $M$ of $m$ vertices on the reduced graph. The solution corresponding to that clique is evaluated according to the selected objective function by applying Equation (5) or (6) and, if better, replaces the best solution found so far. When no clique can be found by procedure *FindClique*, the algorithm terminates, and returns the best solution found overall.

Procedure *Sparsify* applies one of two alternative removal strategies:

- *edge removal*: delete from $E$ the edges with the minimum distance for the *max-minsum DP*

$$E := \left\{ (i,j) \in E : d_{ij} > \min_{(h,k) \in E} d_{hk} \right\}$$

  and those with the minimum and the maximum distance for the *min-diffsum DP*

$$E := \left\{ (i,j) \in E : \min_{(h,k) \in E} d_{hk} < d_{ij} < \max_{(h,k) \in E} d_{hk} \right\}$$

- *vertex removal*: denoting by $\tilde{D}_h$ the sum of the $(m-1)$ largest distances $d_{hk}$, which is an estimate of the unknown aggregate dispersion $D_h$, delete from $N$ the elements with the minimum estimate for the *max-minsum DP*

$$N := \left\{ i \in N : \tilde{D}_i > \min_{h \in N} \tilde{D}_h \right\}$$

  and those with the minimum and the maximum estimate for the *min-diffsum DP*

$$N := \left\{ i \in N : \min_{h \in N} \tilde{D}_h < \tilde{D}_i < \max_{h \in N} \tilde{D}_h \right\}$$

Both strategies remove the less promising components of the graph, in order to drive the search towards cliques which correspond to solutions of higher quality. Notice that procedure *Sparsify* does not necessarily remove elements of the last clique $M$ found by *FindClique*. Thus, $M$ could be found more than once. However, the randomized nature and the diversification mechanisms

built in procedure *FindClique* make this rather unlikely (see Grosso et al. (2008) for details).

The vertex removal strategy recalls a *stingy* heuristic, while the edge removal strategy is inspired by the exact *max-min DP* algorithm proposed by Della Croce et al. (2009). However, both return the best solution found during the process, instead of the last one. Notice that the vertex removal strategy computes $O(n)$ solutions (each step forbids at least one vertex), whereas the edge removal strategy computes $O(n^2)$ solutions (each step forbids at least one edge). Consequently, the latter is slower, but more gradual, and possibly less prone to incorrect choices in its first steps.

### *3.2. Improvement phase*

The improvement phase is a Tabu Search procedure. *Tabu Search* (*TS*) is a well known metaheuristic approach introduced in Glover (1986) to enhance the performance of local search. A complete exposition of Tabu Search can be found in Glover and Laguna (1997). We here focus on our application to the equity-concerned *DP*s.

As already remarked, the feasible solutions of these problems are characterized by their cardinality: they should include exactly $m$ elements. It is therefore a common choice, as well as natural, to define a move as the removal of an element $s \in M$, compensated by the introduction of an element $t \in N \setminus M$. If $M'$ denotes the resulting neighbour solution, we have:

$$M' = M \cup \{t\} \setminus \{s\}$$

Each solution has exactly $m(n-m)$ neighbours. Our Tabu Search procedure visits and evaluates all of them before selecting the incumbent one, i. e., the one which will replace the current solution.

The procedure is described in pseudocode format in Algorithm 2. Drawing inspiration from the basic procedure applied to the *max-sum DP* in Aringhieri et al. (2008), the algorithm divides the possible moves into tabu and nontabu on the basis of two parameters $\ell_{in}$ and $\ell_{out}$. These are denoted as *tabu tenures*, respectively for the insertion and removal of an element. At each iteration, the procedure evaluates all possible moves and returns the nontabu move $(s, t)$ which yields the best solution. An *aspiration criterium* overcomes this general rule, stating that the best move must be returned, even if classified as tabu, if it yields the best solution found so far. The Tabu Search terminates after a specified number of iterations $K$ returning the best solution found in the run.

**Algorithm 2:** *ImproveDP*$\left(M, N, d, m, DP, \ell^m_{\text{in}}, \ell^M_{\text{in}}, \ell^m_{\text{out}}, \ell^M_{\text{out}}, K_w, K_i, K\right)$

**begin**

$f_0 :=$ *EvaluateSolution*$(M, DP)$;
$\tilde{M} := M$; $\tilde{f} := f_0$ ; // Initialize best known solution
$L_i := -\infty$ for all $i \in N$; // Initialize tabu list
$\ell_{\text{in}} := \left(\ell^m_{\text{in}} + \ell^M_{\text{in}}\right)/2$; $\ell_{\text{out}} := \left(\ell^m_{\text{out}} + \ell^M_{\text{out}}\right)/2$ ; // and tabu tenures
**for** $k := 1 \rightarrow K$ **do**
  $(s, t) :=$ *FindBestMove*$(M, N, m, d, L, \ell_{\text{in}}, \ell_{\text{out}}, k, f^*)$;
  $M := M \cup \{t\} \setminus \{s\}$;
  $f_k :=$ *EvaluateSolution*$(M, DP)$;
  **if** *Better*$(f_k, \tilde{f}, DP)$ **then** $\tilde{M} := M$; $\tilde{f} := f_k$;
  $L_s := k$; $L_t := k$; // Update the tabu list
  $\ell_{\text{in}} :=$ *UpdateTabuTenure*$\left(\ell_{\text{in}}, \ell^m_{\text{in}}, \ell^M_{\text{in}}, K_w, K_i, k, f\right)$ ; // and tabu
  $\ell_{\text{out}} :=$ *UpdateTabuTenure*$\left(\ell_{\text{out}}, \ell^m_{\text{out}}, \ell^M_{\text{out}}, K_w, K_i, k, f\right)$ ; // tenures

**Output**: $(\tilde{M}, \tilde{f})$;

*Management of the tabu mechanism.* Our implementation of the tabu mechanism works as follows (Gendreau et al., 1994). A suitable vector $L$ stores for each element $i \in N$ the iteration $L_i$ in which $i$ has joined the current solution (if $i \in M$) or left it (if $i \in N \setminus M$) for the last time. All moves which remove an element $s \in M$ from the solution are labelled as tabu until iteration $L_s + \ell_{\text{out}}$; as well, all moves which introduce an element $t \in N \setminus M$ in the solution are labelled as tabu until iteration $L_t + \ell_{\text{in}}$. At the beginning, all elements of vector $L$ are set to a very large negative value, represented by symbol $-\infty$, so that all moves are nontabu.

To refine the search, intensifying or diversifying it according to its latest results, the values of the two tenures are not fixed once for all, but adaptively tuned from iteration to iteration by procedure *UpdateTabuTenure*, based on the vector $f$ of the values assumed by the objective in the previous iterations. More specifically, the tabu tenure for insertion, $\ell_{in}$, varies within $\left[\ell^m_{in}, \ell^M_{in}\right]$. At the beginning of the algorithm, it is set to the middle point of this range $\left(\ell^m_{in} + \ell^M_{in}\right)/2$; at the $i$-th iteration, if in the $K_w$ previous consecutive iterations the objective $f$ always worsened, $\ell_{in}$ increases by 1 (up to the maximum $\ell^M_{in}$), whilst it decreases by 1 (down to $\ell^m_{in}$) after $K_i$ consecutive improving iterations. The same occurs for $\ell_{out}$, which ranges from $\ell^m_{out}$ to $\ell^M_{out}$, starting at $\left(\ell^m_{out} + \ell^M_{out}\right)/2$ and increasing or decreasing by 1 according to the values

of $f$ in the last iterations. The rationale of this update mechanism is to decrease the tabu tenure if the objective function has steadily improved in the most recent iterations, in order to intensify the search in a region which appears more promising; symmetrically, the tabu tenure increases if the objective function has steadily worsened, in order to diversify the search driving it out of a region which appears less promising.

We set $\ell_{\text{out}} < \ell_{\text{in}}$ because usually $n - m > m$, i. e., more elements are out of the solution than inside it. Consequently, the number of exchanges which remove each element in $M$ is larger than the number of moves which introduce each element out of it. This suggests that the prohibition should last longer for the insertion than for the removal.

*Objective function evaluation and computational complexity.* The most time-consuming operation of procedure *ImproveDP* is, of course, the evaluation of the objective function for each neighbour solution. Both the considered objectives depend on the minimum value of the aggregate dispersions $D_i$ for the elements in $M$; one of them depends on the maximum value (see Equations (5) and (6)). Since a move modifies all the aggregate dispersions, it is in general necessary to compute all indices $D_i$ associated to the $m$ selected elements $i \in M$. A straightforward recomputation of these indices would require $O(m^2)$ time. However, the modified value of each $D_i$ after the removal of $s$ and the insertion of $t$ can be computed in constant time with the following formula:

$$D_i := D_i - d_{is} + d_{it} \tag{7}$$

so that the value of the objective function can be computed in $O(m)$ time for each neighbour solution. Since the number of neighbour solutions evaluated is $m(n-m)$, the overall computational complexity of a single iteration is $O(m^2 n)$.

### *3.3. Diversification mechanism*

Algorithm 3 gives a unified pseudocode of our algorithms. The constructive and improvement phases are repeated a prespecified number of times $T$. At the first application, the constructive phase works on the full set of elements $N$. In the following iterations, on the contrary, it works on a reduced set $N'$ which is obtained forbidding the elements of the best solution found in the previous improvement phase.

This simple mechanism, based on the perturbation of the data, aims to diversify the search, inducing the constructive procedure *BuildDP* to produce

a different starting point $M$ for each iteration of the improvement procedure *ImproveDP*. It is true that the tabu search might actually reintroduce some, or even all, of the original elements, frustrating this aim. However, the reduced set $N'$ moves the search to a solution as different as possible from the original one, therefore limiting as much as possible the probability to replicate it.

Notice that the mechanism actually breaks down when $m > n/2$, because the reduced set $N'$ does not include enough elements to build a full solution. In this case, which never occurs in the instances available in the literature and is less common in practical applications, the mechanism should be modified: instead of removing the elements of $\tilde{M}$ from the data, we should modify procedure *BuildDP* (in particular its subroutines *Sparsify* and *FindClique*) so as to include in the new solution $M$ all the elements of $N \setminus \tilde{M}$. This, in fact, would still guarantee the strongest possible differentiation with respect to the best solution found in the previous phase.

The whole mechanism can also be interpreted as a form of Variable Neighbourhood Search (Hansen and Mladenovic, 2003) with a nonstandard shaking procedure which generates a new starting solution with the maximum number of different elements with respect to the previous one, instead of gradually increasing the difference from iteration to iteration. Such an approach is reasonable if one does not expect to find better solution near the ones already explored.

Diversification is also supported by the random choices of the maximum clique heuristic used in *BuildDP*. Thus, even if different improvement steps find the same solution $\tilde{M}$ and consequently yield the same reduced set $N'$, the following constructive steps will usually build different starting points. Experience on the *max-sum DP* shows that a similar combination of data perturbation and randomness outperforms other more complex algorithmic strategies (Aringhieri and Cordone, 2011).

## 4. Computational results

This section presents our computational experiments on the algorithms described above for the *max-minsum DP* and the *min-diffsum DP*. First we introduce the computational environment, i. e., the machine, the software and the instances used during the experiments (Section 4.1). Then, we discuss the application of a general-purpose *MILP* solver to estimate the largest size of the instances which can be solved to optimality and to provide some reference

**Algorithm 3:** *SolveDP*$\big(N, d, m, DP, T, \ell^m_{\text{in}}, \ell^M_{\text{in}}, \ell^m_{\text{out}}, \ell^M_{\text{out}}, K_w, K_i, K\big)$

**begin**
  $M^* := \emptyset$; $f^* := 0$;
  $N' := N$;
  **for** $t := 1 \to T$ **do**
    $M := \mathit{BuildDP}(N', d, m, DP)$;
    $(\tilde{M}, \tilde{f}) := \mathit{ImproveDP}\big(M, N, d, m, DP, \ell^m_{\text{in}}, \ell^M_{\text{in}}, \ell^m_{\text{out}}, \ell^M_{\text{out}}, K_w, K_i, K\big)$;
    **if** $\mathit{Better}(\tilde{f}, f^*, DP)$ **then** $M^* := \tilde{M}$; $f^* := \tilde{f}$;
    $N' := N \setminus \tilde{M}$;  `// Diversify with respect to iteration` $t$
**Output**: $(M^*, f^*)$;

results to assess the quality of the heuristic algorithms (Section 4.2). The last subsection presents the results of the heuristic algorithms.

### *4.1. Computational environment*

The heuristic algorithms have been coded in C, while the max-clique heuristic of Grosso et al. (2008), which is used as a subroutine in the construction phase, is coded in C++. All of them have been compiled by `gcc` and run on a 2.1 GHz AMD Opteron 8425HE with 12 cores and 16 GB of memory. The *MILP* formulation discussed in Section 4.2 has been implemented via OPL scripts and solved on the same machine. While the *MILP* solver exploits parallel computation, the heuristics are purely sequential.

As anticipated in the introduction, we do not make a comparison with the *GRASP* algorithm of Prokopyev et al. (2009). The first reason for this is that they only provide results for the *max-minsum DP*, ignoring the *min-diffsum DP*. The second is that the range of sizes they consider, i. e., from 30 to 100 elements, is much smaller than the one here taken into account (from 50 to 500 elements). A preliminary phase of experiments showed that the heuristics here proposed can solve in a few seconds instances with $n = 100$, systematically returning the same solution in nearly all runs. The *MILP* solver is never able to improve these solutions, and usually proves their optimality, or at least their proximity to the best achievable bound. A third reason is that the *GRASP* algorithm adopts the same neighbourhood considered in our work, but limits the exploration to at most 100 neighbours and stops as soon as an improving one has been found, according to the so called *first-best* exploration strategy. The computational times reported for 1 000

restarts, but an indeterminate number of local search iterations, range from 5 to 15 seconds on the instances with $n = 100$, running on a 2.66 GHz Intel Core 2 CPU with 3 GB of RAM. As discussed later, our heuristics, running on instances of the same size and on a roughly comparable machine, take a similar time to perform 50 000 local search iterations while exploring the whole neighbourhood, which includes from 900 to 2 400 solutions.

Hence, we decided to skip a detailed comparison with this algorithm and to derive new benchmarks of a larger size from the instances of the *max-sum DP* which are publicly available on the web site of the *Optsicom* project[1] and on our own[2]. The structure of these instances is compatible with any *DP*, as it consists of a set of elements $N$, a distance function $d : N \times N \to \mathbb{R}$ and an integer number $m < |N|$. We consider the following benchmarks, which cover the range of sizes for which significant remarks can be made:

- benchmark APOM, which consists of 40 instances with a number of elements $n$ ranging from 50 to 250, while $m$ is equal to $0.2n$ or $0.4n$; the distance function is Euclidean for 10 instances, random for the other ones;
- benchmark SOM, which consists of 20 instances, with integer-valued distances uniformly extracted at random from $[0, 9]$; $n$ ranges from 100 to 500 and $m$ from $0.1n$ to $0.4n$;
- benchmark GKD, which consists of 20 instances, with Euclidean distances, $n = 500$ and $m = 50$;
- benchmark DM1A, which consists of 20 instances, with real-valued distances uniformly extracted at random from $[0, 10]$, $n = 500$ and $m = 200$;
- benchmark DM1C, which consists of 20 instances, with real-valued distances uniformly extracted at random from $[0, 10]$, $n = 500$ and $m = 50$;
- benchmark DM2, which consists of 20 instances with real-valued distances uniformly extracted at random from $[0, 1000]$, $n = 500$ and $m = 50$.

[1] `http://www.optsicom.es/mdp`

[2] `http://www.di.unito.it/~aringhie/benchmarks.html`

*4.2. Mixed Integer Linear Programming formulations*

This section investigates the application of a general-purpose *MILP* solver, in order to estimate the limit size of the instances which can be solved to optimality without *ad hoc* algorithms, and to assess the quality of the proposed heuristics, at least on the instances for which the mathematical programming bounds remain tight. We remind that, according to previous works, exact methods can solve *max-sum DP* instances up to 100 – 150 elements, adopting semidefinite programming (Aringhieri et al., 2009) or combinatorial techniques (Martì et al., 2010).

The *max-minsum DP* admits the following *MILP* formulation, proposed in Prokopyev et al. (2009), where the binary variables $x_i$ distinguish the elements belonging to the solution ($x_i = 1$) from the other ones ($x_i = 0$) and the continuous variable $\phi$ represents the value of the objective function:

$$\max z = \phi \tag{8a}$$

$$\phi \leq \sum_{j \in N} d_{ij} x_j + Q\,(1 - x_i) \qquad i \in N \tag{8b}$$

$$\sum_{i \in N} x_i = m \tag{8c}$$

$$\phi \in \mathbb{R}, x_i \in \{0, 1\} \qquad i \in N \tag{8d}$$

While constraint (8c) imposes the correct cardinality and constraints (8d) the integrality of the $x_i$ variables, constraints (8b) guarantee that the maximum value of $\phi$ is identical to the minimum aggregate dispersion $D_i$ of the elements belonging to the solution. In fact, thanks to the "big-M" coefficient $Q$, when $x_i = 0$ the right-hand side of the constraint associated to element $i$ becomes much larger than all aggregate dispersions and the constraint is relaxed. To this purpose, $Q$ must overestimate any possible aggregate dispersion in the optimal solution. The original paper sets $Q = 1 + \max_{i \in N} \sum_{j \in N} \max(d_{ij}, 0) - \sum_{j \in N} \min(d_{ij}, 0)$, which can be simplified, with no loss of generality, to $Q = 1 + \max_{i \in N} \sum_{j \in N} d_{ij}$, assuming that $d_{ij} \geq 0$ for all $i, j \in N$.

In our experiments, we strengthen the continuous relaxation of Formulation (8) by minimizing the value of coefficient $Q$, under the condition that no feasible solution should be forbidden. Defining a specific coefficient $Q_i$ for each element $i \in N$ and observing that the aggregate dispersion $D_i$ sums only $m - 1$ distances between $i$ and other elements, we can replace $Q$ in (8b)

with:

$$Q_i = \sum_{j \in N_i^m} d_{ij} \qquad i \in N \tag{9}$$

where $N_i^m$ is composed of the $m-1$ elements with the largest distance from $i$.

The *min-diffsum DP* admits a similar *MILP* formulation, where the binary variables $x_i$ are defined as above, while the continuous variables $\phi$ and $\psi$ represent, respectively, the smallest and the largest aggregate dispersion:

$$\min z = (\psi - \phi) \tag{10a}$$

$$\phi \leq \sum_{j \in N} d_{ij} x_j + Q\,(1 - x_i) \qquad i \in N \tag{10b}$$

$$\psi \geq \sum_{j \in N} d_{ij} x_j - Q'\,(1 - x_i) \qquad i \in N \tag{10c}$$

$$\sum_{i \in N} x_i = m \tag{10d}$$

$$\phi, \psi \in \mathbb{R}, x_i \in \{0, 1\} \qquad i \in N \tag{10e}$$

where $Q$ is the same coefficient used above and $Q'$ relaxes constraint (10c) for all elements $i \in N$ such that $x_i = 0$. The original paper sets $Q' = 1 - \min_{i \in N} \sum_{j \in N} \min(d_{ij}, 0) + \sum_{j \in N} \max(d_{ij}, 0)$, which can be simplified to $Q' = Q = 1 + \max_{i \in N} \sum_{j \in N} d_{ij}$ with the usual nonnegativity assumption. In our experiments, we strengthen the formulation replacing $Q'$ in each constraint with the specific coefficient $Q_i$ defined in (9).

The discussion on the *MILP* formulations focuses on benchmark APOM, because the other benchmarks consist of larger instances, for which no meaningful result can be obtained with a *MILP* solver. Table 1 reports the results obtained with a limit time of 4 hours, i. e., 14 400 seconds on our 12-core parallel machine. The first column provides the number of elements. The following four ones refer to the original setting of $Q$: first, the average percentage gap $(UB - \tilde{z})/\tilde{z}$ between the best value $\tilde{z}$ found by the *MILP* solver and the value $UB$ of the upper bound; then, the average percentage gap $(z^* - \tilde{z})/\tilde{z}$ between the best value $z^*$ found in the whole experimental campaign and that found by the *MILP* solver; then, the number of instances solved to optimality; finally, the average computational time in seconds. The last four columns provide the same information for the formulation strengthened with the $Q_i$ coefficients. It is clear that this strengthening consistently reduces the solving time and the residual gaps, and increases the number of

solved instances. In particular, the largest size for which at least one instance can be solved to optimality grows from $n = 150$ to $n = 200$. Though the improvement is not huge, the reduced coefficients are so easy to compute that their introduction is definitely recommendable. The very limited gap between $\tilde{z}$ and the overall best known result $z^*$ suggests that the use of a general-purpose *MILP* solver is a viable approach, if the computational time is not strictly limited. An additional remark of some interest is that the instances with random distance values distributed in a large range have the largest gaps. This supports the identification of DM2 is the hardest benchmark in our experiments, which will be also confirmed by the results of the heuristic algorithms.

| | Original $Q$ | | | | Strengthened $Q_i$ | | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | $\frac{UB-\tilde{z}}{\tilde{z}}$ | $\frac{z^*-\tilde{z}}{\tilde{z}}$ | #S | CPU | $\frac{UB-\tilde{z}}{\tilde{z}}$ | $\frac{z^*-\tilde{z}}{\tilde{z}}$ | #S | CPU |
| 50 | 0.00% | 0.00% | 8 | 10.14 | 0.00% | 0.00% | 8 | 6.71 |
| 100 | 1.71% | 0.03% | 5 | 6 194.88 | 0.78% | 0.15% | 6 | 4 956.83 |
| 150 | 5.78% | 0.17% | 2 | 11 715.69 | 4.08% | 0.09% | 3 | 10 777.39 |
| 200 | 8.55% | 0.21% | 0 | 14 400.00 | 6.91% | 0.24% | 2 | 12 301.04 |
| 250 | 11.61% | 0.35% | 0 | 14 400.00 | 9.33% | 0.20% | 0 | 14 400.00 |

Table 1: Results obtained with the *MILP* formulation of the *max-minsum DP* on benchmark APOM

The results obtained applying the *MILP* solver to Formulation (10) are quite different. The *MILP* bound, in fact, is very often equal to zero, even after 4 hours of computation, and consequently provides no useful information. Since the problem is a minimization problem, the gap between the *MILP* heuristic value and the overall best known result is expressed as $(\tilde{z} - z^*)/z^*$. Only the smallest instances, up to 50 elements, can be solved to optimality: specifically, the average residual gap is 2.31% and only two instances out of 8 are solved exactly. For the larger instances, the residual gap ranges around $20 - 25\%$, though it does not exhibit a sharp increase with size. As for the structure of the instances, there is a consistent difference between the instances in which the cardinality $m$ is set to $0.2n$ and those with $m = 0.4n$: the former are harder. Rather unexpectedly, the introduction of the reduced coefficients does not bring any improvement. On the contrary, sometimes it yields worse results, probably because the information provided by the *MILP* relaxation is scarcely useful. In fact, the heuristics here proposed clearly out-

perform the *MILP* solver, even neglecting the fact that the computational time they require is orders of magnitude smaller. The direct use of a general-purpose solver on the *min-diffsum DP* is, therefore, almost ineffective, and specific heuristics must be applied to obtain solutions of acceptable quality. On the other hand, part of this huge difference between the percentage gaps obtained on the *max-minsum DP* and the *min-diffsum DP* is due to the fact that the optimal objective value for the former is typically large, being an aggregate dispersion, whereas it can be very small (potentially, even zero) for the latter.

### *4.3. Experiments on the proposed heuristics*

The algorithms proposed in this paper iteratively apply a constructive procedure and an improvement procedure, with an auxiliary diversification mechanism. This section first compares the quality of the solutions produced by the two constructive procedures. Then, it discusses the tuning of the Tabu Search parameters and of the total number of iterations which according to experience are required to obtain stable results. Finally, it investigates the impact on the final result of the restart frequency and of the constructive procedure adopted. For comparison purposes, we also test a simple random initialization, verifying that, in practice, the use of a refined constructive procedure is not justified since the advantage of a good initialization fades away after a sufficient number of improvement steps. For the sake of briefness, the preliminary phases of this analysis focus on a restricted set of benchmarks, namely APOM, which includes the smallest instances, and DM2, which includes the hardest ones.

*Constructive procedures.* We first applied the edge removal and the vertex removal strategies as standalone algorithms, to estimate the quality of the solutions they can provide to a subsequent improvement procedure.

The first two columns of Table 2 report the size of the instances considered for the *max-minsum DP*: those with $n$ ranging from 50 to 250 belong to benchmark APOM (8 for each size), whereas those with $n = 500$ belong to benchmark DM2 (20 overall). The horizontal line stresses the separation between the two benchmarks. The following two pairs of columns report, for each initialization procedure, the average computational time in seconds and the average percentage gap with respect to the best result obtained in the whole experimental campaign. This is assumed as the best possible approximation for the optimum, since the lower bounds yielded by the *MILP*

formulations are very weak. Table 3 provides the same information for the *min-diffsum DP*. As expected, the edge removal strategy outperforms the vertex removal strategy but requires a longer computational time. The quality of both procedures is, however, modest.

The main interest of these results, then, concerns the different performance on the two benchmarks and on the two problems. Benchmark DM2, in fact, exhibits smaller gaps and a stronger difference between the computational times of the two methods. This is because the distance function has a wider range of values, so that the edge removal strategy removes less edges at each step and evaluates more solutions overall.

| | | Edge removal | | Vertex removal | |
|---|---|---|---|---|---|
| $n$ | $m$ | CPU | Gap | CPU | Gap |
| 50 | { 10,20 } | 0.12 | 14.76% | 0.00 | 59.88% |
| 100 | { 20,40 } | 0.20 | 21.92% | 0.02 | 59.79% |
| 150 | { 30,60 } | 0.43 | 24.71% | 0.03 | 58.92% |
| 200 | { 40,80 } | 0.86 | 27.45% | 0.09 | 58.87% |
| 250 | { 50,100 } | 1.74 | 29.85% | 0.18 | 56.63% |
| 500 | 50 | 43.45 | 14.34% | 1.59 | 32.70% |

Table 2: Computational time and percentage gap with respect to the best known solution of the two initialization procedures on the *max-minsum DP*

Table 3 reports the corresponding results for the *min-diffsum DP*: they exhibit the same dependences, but much larger gaps, thus confirming that this problem is harder.

*Tabu Search parameters.* The improvement procedure is a Tabu Search algorithm with two adaptively varying tabu tenures, one for the insertion and one for the removal of elements. After a preliminary phase of experiments, we tuned the range of the former tenure, $\ell_{\text{in}}$, as $[8; 14]$: at first, the tenure is set equal to 11, which is the middle point of the range; then, it decreases after $K_i = 3$ consecutive improving iterations and increases after $K_w = 5$ consecutive worsening iterations. The range of the tenure for removal, $\ell_{\text{out}}$, is $[3; 7]$: its starting value is 5 and it is updated with the same rule of the other tenure. These values confirm the general remark we made on the *max-sum DP* (Aringhieri et al., 2008; Aringhieri and Cordone, 2011) that the tabu tenure for insertion should be larger than that for removal, because the number of elements out of the solution exceeds the number of those inside it.

| | | Edge removal | | Vertex removal | |
|---|---|---|---|---|---|
| $n$ | $m$ | CPU | Gap | CPU | Gap |
| 50 | { 10,20 } | 0.10 | 370.86% | 0.00 | 609.38% |
| 100 | { 20,40 } | 0.13 | 514.62% | 0.01 | 477.68% |
| 150 | { 30,60 } | 0.21 | 614.71% | 0.03 | 474.41% |
| 200 | { 40,80 } | 0.29 | 687.18% | 0.05 | 408.52% |
| 250 | { 50,100 } | 0.41 | 798.39% | 0.11 | 378.45% |
| 500 | 50 | 22.29 | 341.31% | 0.84 | 441.03% |

Table 3: Computational time and percentage gap with respect to the best known solution of the two initialization procedures on the *min-diffsum DP*

*Total number of iterations.* We then experimentally identified the number of iterations after which the objective function tends, for all the available instances, to converge to a stable value with only occasional spaced out improvements. This occurs between 10 000 and 50 000 iterations for the *max-minsum DP* and between 50 000 and 100 000 iterations for the *min-diffsum DP*. On the basis of these experiments, in the following we decided to set the total number of local search iterations to 50 000 for the *max-minsum DP* and 100 000 iterations for the *min-diffsum DP*. This is approximately twice the average number of iterations $k^*$ after which the heuristic finds the last improving solution on the benchmarks considered (see Table 4).

| | | *max-minsum DP* | | *min-diffsum DP* | |
|---|---|---|---|---|---|
| $n$ | $m$ | Edge removal | Vertex removal | Edge removal | Vertex removal |
| 50 | { 10,20 } | 5132.5 | 3837.1 | 4197.5 | 7992.8 |
| 100 | { 20,40 } | 17647.3 | 10053.8 | 39028.5 | 53817.3 |
| 150 | { 30,60 } | 15107.3 | 14449.9 | 37091.3 | 50682.9 |
| 200 | { 40,80 } | 26350.0 | 15733.9 | 66693.3 | 53613.3 |
| 250 | { 50,100 } | 27673.0 | 27838.1 | 67624.9 | 29355.6 |
| 500 | 50 | 23601.0 | 26310.8 | 65936.1 | 59901.0 |

Table 4: Average number of iterations before the last improvement with the two variants of the proposed heuristic on the *max-minsum DP* ($T = 1$ and $K = 50\,000$) and the *min-diffsum DP* ($T = 1$ and $K = 100\,000$)

*Restart frequency.* The following phase of experiments focused on investigating whether it is more profitable to concentrate the improvement iterations in a single run of the Tabu Search procedure or to distribute them among independent runs. We kept the same total number of iterations determined above, i. e., 50 000 for the *max-minsum DP* and 100 000 for the *min-diffsum DP* and divided it into $T$ blocks of $K$ iterations each. At the beginning of each block, the constructive procedure builds a new starting solution, operating on the auxiliary graph defined by the diversification mechanism. The number of restarts $T$ and the number of local search iterations per restart $K$ are, therefore, inversely proportional.

This experiment also allows to investigate whether the constructive procedure exerts or not a lasting influence on the final result. In fact, a refined initialization is justified only if the time spent to perform it is compensated by a significantly better final solution. Otherwise, the same time could be employed to increase the number of improvement iterations. In order to evaluate the long-term influence of the starting solution on the final result, we have applied the same total number of iterations $K\,T$ to the solutions produced by the edge removal strategy, by the vertex removal strategy and by a simple random initialization, denoted in the following as *random strategy*. If the final result exhibits little or no dependence on the initialization, we will be authorized to prefer faster and simpler mechanisms. If, on the contrary, a better starting solution is related to a better final result, we will need to investigate further the balance between the time spent in the constructive and the improvement phase.

Starting, as usual, with the *max-minsum DP*, Figures 1 and 2 show, respectively for benchmark APOM and benchmark DM2, the average percentage gap $(z^* - z)\,/z$ between the best value $z^*$ found in the whole experimental campaign and the value found by the heuristics. For each strategy, we test the following restart frequencies: $T = 1$, 5, 10, 20, 50 or 100 restarts, which correspond to $K = 50\,000$, $10\,000$, $5\,000$, $2\,500$, $1\,000$ and 500 local search iterations for each restart.

Summarising, when applied to the *max-minsum DP*, all of the three strategies provide solutions very close to the best known ones, and probably also to the optimum (though this can be guaranteed only for $n \leq 100$).

To estimate in detail the sensitivity of the algorithm performance to the restart frequency, we have applied *Wilcoxon's matched-pairs signed-ranks test* Wilcoxon (1945). With respect to the *random strategy*, the test indicates that the setting with $T = 10$ restarts with $K = 5\,000$ iterations each, which

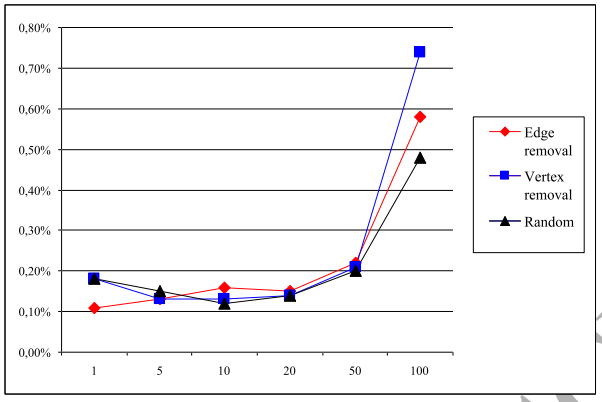


Figure 1: Average percentage gap with respect to the best known result obtained with the vertex removal strategy, the edge removal strategy and a random restart, applied with different restart frequencies to benchmark APOM for the *max-minsum DP*

is the best on average, is not significantly different from $T = 5$ and $T = 20$ (i. e., the probability that the difference is produced by random fluctuations exceeds 5%), but it dominates the other restart frequencies considered. For the *vertex removal strategy*, the best setting is $T = 5$, but the other settings with $T \leq 50$ are not significantly worse. For the *edge removal strategy*, the best setting is $T = 1$, but the settings with $T \leq 20$ are not significantly worse. These results suggest that, while a random initialization has a specific range of effective restart frequencies, a good initialization allows the algorithm to perform well with less restarts, and to be more robust with respect to the restart frequency.

The results on benchmark DM2 partly confirm these indications. In fact, the *random strategy* performs best with $T = 5$ restarts, but $T = 1$ and $T = 10$ are not significantly worse. So, there is once again an optimal range of values, largely overlapping with the previous one. The *vertex removal strategy* and the *edge removal strategy* perform best with $T = 20$, but all settings with $T \leq 50$ have similar performance. Thus, a good initialization still improves the robustness of the heuristic, and the optimal range of parameter values is more or less the same. The fact that the parameter setting which performs best has a larger value of $T$ could be related to the larger size of these instances, but this does not hold for the random strategy.

Similar comments can be made on the other benchmarks, on which we

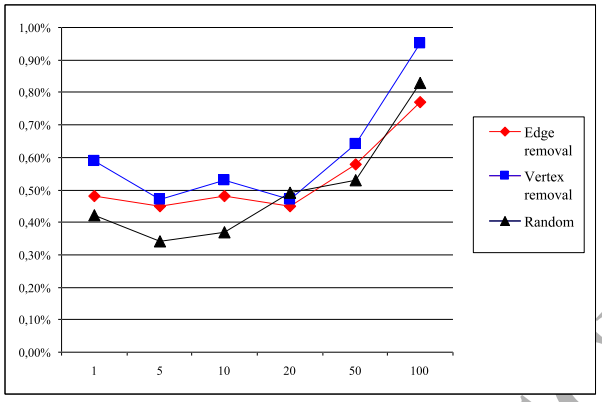


Figure 2: Average percentage gap with respect to the best known result obtained with the vertex removal strategy, the edge removal strategy and a random restart, applied with different restart frequencies to benchmark DM2 for the *max-minsum DP*

give here only summarised comments for the sake of briefness. The Euclidean benchmark GKD exhibits very small gaps (always $< 0.14\%$) with nearly no dependence on the value of $T$. For the other three benchmarks, the best value for $T$ varies, but there is always a range of statistically equivalent values which never includes the setting $T = 100$ and often excludes $T = 50$. The range for the random initialization, in particular, tends to be smaller and usually excludes also $T = 1$.

Figures 3 and 4 show the corresponding information for the *min-diffsum DP* with $KT = 100\,000$. The values considered for the number of restarts $T$ are $T = 1$, 10, 20, 40, 100 or 200, and the corresponding number of local search iterations for each restart are $K = 100\,000$, $10\,000$, $5\,000$, $2\,500$, $1\,000$ and 500. The gap here is expressed as $(z - z^*)/z^*$, because this is a minimization problem. In summary, all of the three strategies yield a $5-8\%$ gap with respect to the best known result. This gap is not huge, but it is one order of magnitude larger than that observed on the *max-minsum DP*, confirming that the *min-diffsum DP* is harder. Moreover, the best known result is also less likely to be optimal.

The *random strategy* performs best with $T = 40$ restarts and $K = 2\,500$ iterations each, but lower frequencies are not significantly worse. The *vertex removal strategy* performs best with $T = 20$, but the setting with $T = 10$ is statistically equivalent. The *edge removal strategy*, in the end, performs best

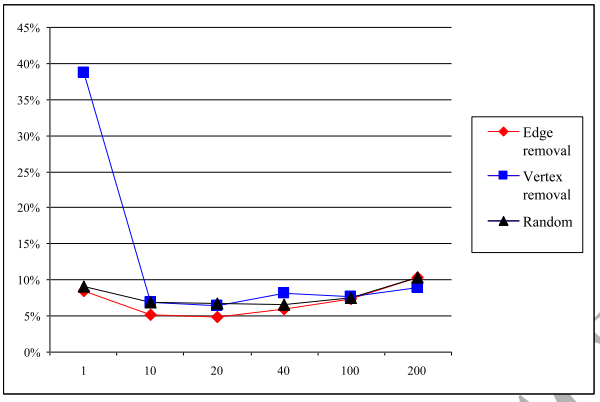


Figure 3: Average percentage gap with respect to the best known result obtained with the vertex removal strategy, the edge removal strategy and a random restart, applied with different restart frequencies, on benchmark APOM for the *min-diffsum DP*

with $T = 20$, though all settings with $T \leq 40$ are statistically equivalent.

The results on benchmark DM2 (Figure 4) with the *random strategy* show an unexpected minimum gap with $T = 1$ restart, with a maximum in $T = 10$ followed by steady improvements as $T$ increases. Wilcoxon's test suggests that the results obtained with $T \geq 20$ are not statistically different from those obtained with $T = 1$. The *vertex removal strategy* and the *edge removal strategy* also perform best with many restarts ($T = 200$), though, once again, Wilcoxon's test suggests that the difference with respect to the other settings is not statistically dominated. So, there is a slightly improving trend with higher restart frequencies, but the strength of this trend is not statistically significant. The other benchmarks tend to confirm the remarks made on APOM, rather than those made on DM2: the three strategies very often perform best with $T = 10$, while the gap tends to rise with more frequent restarts. The increase is slow on benchmark SOM, for which many different settings are statistically equivalent, whereas it is rather quick on the other benchmarks.

*Comparison of the initialization procedures.* The experiments on the restart frequency also suggest that none of the three initialization strategies dominates the others. The random strategy is often slightly better on average, but it seems to be less robust, since its optimal range of frequencies tends to be smaller. Indeed, we have also applied Wilcoxon's test to compare the

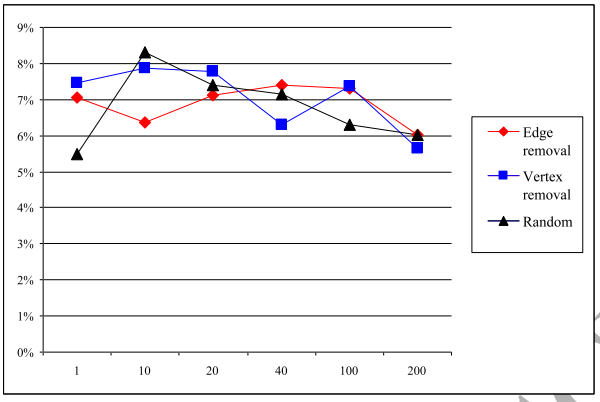


Figure 4: Average percentage gap with respect to the best known result obtained with the vertex removal strategy, the edge removal strategy and a random restart, applied with different restart frequencies, on benchmark DM2 for the *min-diffsum DP*

three strategies, each one with the best setting of the restart frequency, on the overall collection of benchmark instances. The test fail to reveal any statistically significant difference.

A complementary point of view can be found in Table 5, which reports, for both problems (first column) and all benchmark classes (second column), the total number of instances (third column) and the number of instances on which each of the three reinitialization procedures hits the best known result (last three columns). With respect to this index, none of the alternatives outperforms the others, though the edge removal strategy tends to find more best results than the other two.

Summarising, there is no statistical dominance among the three heuristics considered. The random initialization is much faster, especially on large instances, and it has the nonnegligible advantage of being simpler. Though it is less robust with respect to the restart frequency, this parameter can vary within a reasonably wide range without significantly affecting the quality of the final solution (for example, setting the number of restarts to $T = 10$ guarantees an effective performance on all benchmark classes). Our conclusion is that it is reasonable to prefer a simple random initialization, since it allows to gain additional time, which could be exploited to increase the number of improvement iterations.

| Problem | Benchmark | # of inst. | Edge removal | Vertex removal | Random |
|---|---|---|---|---|---|
| *max-minsum DP* | APOM | 40 | 28 | 26 | 29 |
| | SOM | 20 | 11 | 15 | 10 |
| | GKD | 20 | 15 | 15 | 12 |
| | DM1A | 20 | 9 | 5 | 6 |
| | DM1C | 20 | 11 | 5 | 4 |
| | DM2 | 20 | 10 | 5 | 5 |
| | Tot. | 140 | 84 | 71 | 66 |
| *min-diffsum DP* | APOM | 40 | 24 | 20 | 18 |
| | SOM | 20 | 11 | 11 | 12 |
| | GKD | 20 | 7 | 9 | 5 |
| | DM1A | 20 | 10 | 8 | 2 |
| | DM1C | 20 | 5 | 8 | 7 |
| | DM2 | 20 | 5 | 9 | 6 |
| | Tot. | 140 | 62 | 65 | 50 |

Table 5: Number of best known results reached by applying each initialization procedure for each instance class of the *max-minsum DP* and the *min-diffsum DP*

*Average gap and computational time of the best parameter setting.* To confirm these conclusions, we have adopted the random initialization strategy and compared its performance to the best results obtained in the whole experimental campaign. We keep the total number of improvement iterations to $KT = 50\,000$ for the *max-minsum DP* and to $KT = 100\,000$ for the *min-diffsum DP*. We set the number of restarts to $T = 10$, so that each improvement phase consists of $K = 5\,000$ iterations for the *max-minsum DP* and to $K = 10\,000$ iterations for the *min-diffsum DP*.

Table 6 reports, for both problems (first column) and for each class of benchmark instances (second column), the number of instances (third column), the average percentage gap obtained with this setting with respect to the best known results (fourth column) and the average CPU time in seconds required (fifth column).

The first remark is that the percentage gap obtained for the *max-minsum DP* is small (nearly always $< 1\%$), while it is still significant for the *min-diffsum DP* (between 3 and 10% on average). The computational times are in good accordance with the theoretical analysis of Section 3.2. In fact, they are approximately proportional to $m^2\,(n-m)$, and the time required to solve the *max-minsum DP* is half that required for the *min-diffsum DP*, given that the number of local search iterations is also half as much.

| Problem | Benchmark | # of inst. | Avg. gap | Avg. CPU |
|---|---|---|---|---|
| *max-minsum DP* | APOM | 40 | 0.12% | 108.38 |
| | SOM | 20 | 0.58% | 828.73 |
| | GKD | 20 | 0.03% | 342.67 |
| | DM1A | 20 | 0.12% | 5 042.17 |
| | DM1C | 20 | 0.16% | 368.14 |
| | DM2 | 20 | 0.37% | 370.23 |
| | Avg. | - | 0.21% | 1 024.10 |
| *min-diffsum DP* | APOM | 40 | 6.92% | 212.39 |
| | SOM | 20 | 10.66% | 1 707.06 |
| | GKD | 20 | 4.78% | 714.55 |
| | DM1A | 20 | 3.20% | 10 452.93 |
| | DM1C | 20 | 10.16% | 767.71 |
| | DM2 | 20 | 8.31% | 768.09 |
| | Avg. | - | 7.28% | 2 119.30 |

Table 6: Average percentage gap with respect to the best known results and average CPU time in seconds for each instance class of the *max-minsum DP* and the *min-diffsum DP* with a random inititialization, and $T = 10$ restarts.

For reference purposes, we report in the Appendix our best known results for each instance of the available benchmarks.

## 5. Conclusions

We have proposed constructive and improvement heuristics for two equity-concerned *DP*s, namely the *max-minsum DP* and the *min-diffsum DP*, based on the fundamental ideas which underly the state-of-the-art algorithms on efficiency-concerned *DP*s.

The investigation of smart techniques to reduce the computational complexity of each iteration has allowed a shift from the straightforward quadratic evaluation to a linear update mechanism. In this way, it is possible to solve in a matter of few minutes instances up to 500 elements, much larger than the ones previously considered in the literature. Ongoing work is devoted to devise auxiliary data structure or different neighbourhoods, which could improve the average case complexity, possibly trading a loss in the quality of the solution for a reduction of the computational time. In this regard, the equity-concerned *DP*s appear intrinsically challenging, due to the hybrid nature of their objective function, which combines two different operators: a

sum on the lower level to compute the $D_i$ coefficients, and a minimization or maximization at the upper level. Exchanging each pair of elements modifies each aggregate dispersion by an independent amount (see Equation (7)). This makes unlikely that the worst-case time to update the minimum (and the maximum) aggregate dispersion could become less than linear.

The comparison of different initialization procedures and different restart frequencies suggests that the best approach for these problems is to apply a simple random initialization, as it is for the *max-sum DP* and contrary to the *max-min DP*. However, the best restart frequency appears to be larger than that reported for the *max-sum DP* in Brimberg et al. (2009) and Aringhieri and Cordone (2011), and more specific for each class of instances. Moreover, while the *max-minsum DP* appears to be easily solved to near optimality, the *min-diffsum DP* exhibits larger gaps and much less stable results. In other words, these problems partly confirm and partly disprove what was known for the more investigated efficiency-concerned *DP*s.

We have also estimated the practicability of directly applying a general-purpose solver to a *MILP* formulation of the two problems, obtaining very different outcomes. On one hand, the *max-minsum DP* can be solved exactly up to $n = 150$ elements, obtaining limited gaps and solutions comparable to those provided by tailored heuristics for larger instances, up to $n = 500$ elements. On the other hand, the general-purpose solver is unable to solve even small instances of the *min-diffsum DP* ($n = 50$), and it does not provide useful lower bounds.

## References

## References

Adil, G. K., Ghosh, J. B., 2005. Maximum diversity/similarity models with extension to part grouping. International Transactions in Operations Research 12, 311–323.

Aringhieri, R., 2009. Composing medical crews with Equity and Efficiency. Central European Journal of Operations Research 17 (3), 343–357.

Aringhieri, R., Bruglieri, M., Cordone, R., 2009. Optimal results and tight bounds for the maximum diversity problem. Foundations of Computing and Decision Sciences 34 (2), 73–85.

Aringhieri, R., Cordone, R., Feb. 2011. Comparing local search metaheuristics for the maximum diversity problem. Journal of the Operational Research Society 62 (2), 266–280.

Aringhieri, R., Cordone, R., Melzani, Y., 2008. Tabu search vs. GRASP for the maximum diversity problem. 4OR: A Quarterly Journal of Operations Research 6 (1), 45–60.

Brimberg, J., Mladenovic, N., Urosevic, D., Ngai, E., 2009. Variable neighborhood search for the heaviest $k$-subgraph. Computers & Operations Research 36 (11), 2885–2891.

Della Croce , F., Grosso, A., Locatelli, M., 2009. A heuristic approach for the max-min diversity problem based on max-clique. Computers & Operations Research 36 (8), 2429–2433.

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

Erkut, E., 1990. The discrete $p$-dispersion problem. European Journal of Operational Research 46 (1), 48–60.

Erkut, E., Neuman, S., 1989. Analytical models for locating undesirable facilities. European Journal of Operational Research 40 (3), 275–291.

Erkut, E., Neuman, S., 1991. Comparison of four models for dispersing facilities. Information Systems and Operational Research (INFOR) 29 (2), 68–86.

Gallego, M., Duarte, A., Laguna, M., Martì, R., Dec. 2009. Hybrid heuristics for the maximum diversity problem. Computational Optimization and Applications 44 (3), 411–426.

Gendreau, M., Hertz, A., Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. Management Science 40 (10), 1276–1290.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

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.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers.

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.

Hansen, P., Mladenovic, N., 2003. Variable neighborhood search. In: Glover, F., Kochenagen, G. (Eds.), Handbook of Metaheuristics. Kluwer Academic Publishers, pp. 145–184.

Kuby, M. J., 1987. Programming models for facility dispersion: The $p$-dispersion and maximum dispersion problems. Geographical Analysis 19, 315–329.

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

Martì, R., Gallego, M., Duarte, A., 2010. A branch and bound algorithm for the maximum diversity problem. European Journal of Operational Research 200, 36–44.

Martì, R., Gallego, M., Duarte, A., Pardo, E., 2011. Heuristics and metaheuristics for the maximum diversity problem. Journal of Heuristics, 1–25.

Martí, R., Sandoya, F., 2012. GRASP and path relinking for the equitable dispersion problem. Computers & Operations Research[in press, DOI:10.1016/j.cor.2012.04.005].

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

Pisinger, D., 2006. Upper bounds and exact algorithms for p-dispersion problems. Computers and Operations Research 33 (5), 1380–1398.

Porumbel, D. C., Hao, J.-K., Glover, F., 2011. A simple and effective algorithm for the maxmin diversity problem. Annals of Operations Research 186, 275–293.

Prokopyev, O. A., Kong, N., Martinez-Torres, D. L., 2009. The equitable dispersion problem. European Journal of Operational Research 197 (1), 59–67.

Resende, M. G. C., Martì, R., Gallego, M., Duarte, A., 2010. GRASP and path relinking for the max-min diversity problem. Computers & Operations Research 37 (3).

Silva, G. C., de Andrade, M. R. Q., Ochi, L. S., Martins, S. L., Plastino, A., 2007. New heuristics for the maximum diversity problem. Journal of Heuristics 13, 315–336.

Stadler, P. F., 1992. Correlation in landscapes of combinatorial optimization problems. Europhysics Letters 20, 479–482.

Wang, J., Zhou, Y., Cai, Y., Yin, J., Apr. 2012. Learnable tabu search guided by estimation of distribution for maximum diversity problems. Soft Computing 16 (4), 711–728.

Weitz, R. R., Lakshminarayanan, S., 1998. An empirical comparison of heuristic methods for creating maximally diverse groups. The Journal of the Operational Research Society 49, 635–646.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics 1, 80–83.

Wu, Q., Hao, J.-K., 2013. A hybrid metaheuristic method for the maximum diversity problem. European Journal of Operational Research[in press, DOI:10.1016/j.ejor.2013.06.002].

## Appendix A. Best known results

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 01a050m10 | 95.071 | 1.406 | 21c050m10 | 58 994 | 1 124 |
| 02a050m20 | 184.495 | 14.721 | 22c050m20 | 91 338 | 6 205 |
| 03a100m20 | 195.300 | 3.645 | 23c100m20 | 112 002 | 2 135 |
| 04a100m40 | 370.508 | 25.497 | 24c100m40 | 194 357 | 11 098 |
| 05a150m30 | 294.565 | 6.562 | 25c150m30 | 164 357 | 3 849 |
| 06a150m60 | 556.147 | 46.987 | 26c150m60 | 290 121 | 13 087 |
| 07a200m40 | 392.625 | 11.744 | 27c200m40 | 217 583 | 5 604 |
| 08a200m80 | 738.100 | 63.557 | 28c200m80 | 391 587 | 19 714 |
| 09a250m50 | 489.318 | 14.821 | 29c250m50 | 270 932 | 6 561 |
| 10a250m100 | 921.230 | 82.434 | 30c250m100 | 485 728 | 22 889 |
| 11b050m10 | 61 831 | 1 091 | 31d050m10 | 74 113 | 1 049 |
| 12b050m20 | 108 248 | 5 552 | 32d050m20 | 145 411 | 4 564 |
| 13b100m20 | 118 380 | 3 634 | 33d100m20 | 152 236 | 2 529 |
| 14b100m40 | 214 671 | 10 803 | 34d100m40 | 294 991 | 8 979 |
| 15b150m30 | 171 973 | 6 547 | 35d150m30 | 228 316 | 3 950 |
| 16b150m60 | 315 019 | 13 793 | 36d150m60 | 443 413 | 13 127 |
| 17b200m40 | 224 560 | 8 179 | 37d200m40 | 303 932 | 5 401 |
| 18b200m80 | 416 971 | 19 939 | 38d200m80 | 590 671 | 19 184 |
| 19b250m50 | 278 178 | 12 152 | 39d250m50 | 379 314 | 6 608 |
| 20b250m100 | 519 758 | 23 338 | 40d250m100 | 741 573 | 21 511 |

Table A.7: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark APOM

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 1 | 29 158.15 | 1 120.68 | 11 | 28 984.71 | 1 114.49 |
| 2 | 29 073.30 | 1 088.45 | 12 | 28 919.71 | 1 079.26 |
| 3 | 29 086.14 | 1 045.53 | 13 | 29 083.33 | 1 054.02 |
| 4 | 29 093.40 | 1 090.29 | 14 | 29 218.03 | 1 099.61 |
| 5 | 28 924.19 | 1 049.80 | 15 | 29 343.22 | 1 073.46 |
| 6 | 29 093.21 | 1 002.56 | 16 | 29 135.51 | 1 107.80 |
| 7 | 29 185.38 | 1 098.34 | 17 | 28 993.85 | 1 084.23 |
| 8 | 29 156.69 | 1 117.33 | 18 | 29 015.67 | 1 094.80 |
| 9 | 28 967.58 | 1 079.44 | 19 | 29 189.34 | 1 111.04 |
| 10 | 28 971.87 | 1 132.51 | 20 | 29 132.29 | 1 087.97 |

Table A.8: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark DM2

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 01-n100m10 | 62 | 0 | 11-n300m90 | 433 | 21 |
| 02-n100m20 | 110 | 5 | 12-n300m120 | 565 | 30 |
| 03-n100m30 | 150 | 8 | 13-n400m40 | 215 | 9 |
| 04-n100m40 | 193 | 10 | 14-n400m80 | 396 | 18 |
| 05-n200m20 | 115 | 4 | 15-n400m120 | 570 | 27 |
| 06-n200m40 | 207 | 9 | 16-n400m160 | 743 | 36 |
| 07-n200m60 | 293 | 15 | 17-n500m50 | 262 | 11 |
| 08-n200m80 | 381 | 21 | 18-n500m100 | 492 | 22 |
| 09-n300m30 | 165 | 7 | 19-n500m150 | 713 | 33 |
| 10-n300m60 | 301 | 13 | 20-n500m200 | 926 | 44 |

Table A.9: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark SOM

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 1 | 761.98 | 8.02 | 11 | 765.74 | 7.67 |
| 2 | 771.15 | 8.17 | 12 | 754.50 | 8.01 |
| 3 | 764.02 | 7.62 | 13 | 755.97 | 7.46 |
| 4 | 763.81 | 7.69 | 14 | 760.16 | 7.66 |
| 5 | 765.63 | 8.30 | 15 | 757.41 | 8.15 |
| 6 | 761.34 | 8.11 | 16 | 769.81 | 8.08 |
| 7 | 764.68 | 8.48 | 17 | 756.73 | 7.79 |
| 8 | 766.07 | 7.87 | 18 | 759.64 | 8.28 |
| 9 | 755.15 | 8.10 | 19 | 761.52 | 7.81 |
| 10 | 769.00 | 8.52 | 20 | 763.97 | 8.33 |

Table A.10: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark GKD

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 1 | 1 034.08 | 41.07 | 11 | 1 030.54 | 41.59 |
| 2 | 1 031.22 | 41.58 | 12 | 1 028.18 | 40.78 |
| 3 | 1 031.47 | 41.43 | 13 | 1 036.35 | 41.58 |
| 4 | 1 029.77 | 41.19 | 14 | 1 032.48 | 41.71 |
| 5 | 1 028.80 | 41.95 | 15 | 1 029.48 | 40.68 |
| 6 | 1 031.37 | 40.75 | 16 | 1 033.87 | 42.25 |
| 7 | 1 029.56 | 42.22 | 17 | 1 032.40 | 41.06 |
| 8 | 1 028.46 | 41.03 | 18 | 1 028.28 | 41.72 |
| 9 | 1 032.64 | 41.19 | 19 | 1 030.34 | 42.32 |
| 10 | 1 030.37 | 41.10 | 20 | 1 030.55 | 41.10 |

Table A.11: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark DMI1A

| Instance | *max-minsum* | *min-diffsum* | Instance | *max-minsum* | *min-diffsum* |
|---|---|---|---|---|---|
| 1 | 291.04 | 11.39 | 11 | 291.63 | 10.13 |
| 2 | 289.92 | 10.98 | 12 | 290.05 | 10.72 |
| 3 | 292.31 | 10.88 | 13 | 292.82 | 10.37 |
| 4 | 289.66 | 10.90 | 14 | 291.98 | 9.46 |
| 5 | 289.60 | 10.87 | 15 | 291.62 | 10.50 |
| 6 | 292.67 | 10.43 | 16 | 290.69 | 9.77 |
| 7 | 290.15 | 10.60 | 17 | 290.90 | 11.19 |
| 8 | 290.69 | 10.95 | 18 | 289.63 | 10.83 |
| 9 | 292.64 | 10.75 | 19 | 290.17 | 10.86 |
| 10 | 291.78 | 10.92 | 20 | 289.86 | 10.98 |

Table A.12: Best known results for the *max-minsum DP* and *min-diffsum DP* instances of benchmark DM1C