Heuristics for the Bi-Objective Diversity Problem

J. M. Colmenar[a], R. Martí[b], A. Duarte[a]

[a] *Dept. Computer Sciences, Universidad Rey Juan Carlos, Móstoles, 28933 (Madrid), Spain*
[b] *Dept. de Estadística e Investigación Operativa, Universidad de Valencia, Burjassot, 46100 (Valencia), Spain*

Abstract

The Max-Sum diversity and the Max-Min diversity are two well-known optimization models to capture the notion of selecting a subset of diverse points from a given set. The resolution of their associated optimization problems provides solutions of different structures, in both cases with desirable characteristics. They have been extensively studied and we can find many metaheuristic methodologies, such as Greedy Randomized Adaptive Search Procedure, Tabu Search, Iterated Greedy, Variable Neighborhood Search, and Genetic algorithms applied to them to obtain high quality solutions. In this paper we solve the bi-objective problem in which both models are simultaneously optimized. No previous effort has been devoted to study the "combined problem" from a multi-objective perspective. In particular, we adapt the mono-objective methodologies applied to this problem to the resolution of the bi-objective problem, obtaining approximations to its efficient front. An empirical comparison discloses the best alternative to tackle this $\mathcal{NP}$-hard problem.

Keywords: Diversity problems, multi-objective, metaheuristics, optimization

1. Introduction

The problem of maximizing diversity refers to the selection of a subset of elements from a given set in such a way that the diversity among the selected elements is maximized (Glover et al., 1995). We can find different applications of maximizing diversity in real-world situations. For instance, many universities in the United States, when determining admission policies, go beyond selecting through the academic grades, and also consider other factors in the search for a diverse set of students (Ramirez, 1979). In market planning, it is often desirable to maximize both, the number and the diversity of forces in a brand profile (Keely, 1989). Other contexts in which maximizing the diversity can be applied include plant cultivation (Porter et al., 1975), social problems (Swierenga, 1977), immigration policies that promote ethnic diversity (McConnell, 1988),

Email addresses: josemanuel.colmenar@urjc.es (J. M. Colmenar), rafael.marti@uv.es (R. Martí), abraham.duarte@urjc.es (A. Duarte)

ecological preservation (Unkel, 1985), design of products (von Ghyezy, 1986), task-fore management (Thomas, 1990), curriculum design (Jackson, 1991), and management of genetic resources (Glover, 1992).

As other optimization problems, in spite of its simplicity, it is a challenge even for modern solving techniques. As a matter of fact, the first obstacle that we encounter when dealing with diversity is its modeling. Note that when we talk about diversity, we are assuming the existence of a distance function in the space where elements belong. Distance functions, such as the well-known Euclidean distance, typically compute a value to measure the similarity or proximity of two elements. When we consider a subset of more than two elements, we have a distance value for each pair of elements in the subset, and we have to compute a single diversity value from all of them. In this way, we can compare two different subsets in our search for the best one. This is where the mathematical model plays a key role capturing the notion of diversity by specifying how to compute a single diversity value from many pairwise distances. A few models, and many solving methods have been proposed in the last few years.

The most studied model is probably the *Maximum Diversity Problem* (MDP) also known as the *Max-Sum Diversity Model* (Ghosh, 1996), in which the sum of the distances between the selected elements is maximized. Considering that n is the number of elements in the original set, and m the number of selected elements (the subset cardinality), we can formulate it in mathematical terms as:

$$\begin{array}{lll} \text{(MDP) Maximize} & z_{MS}(x) = \sum\limits_{i<j} d_{ij} x_i x_j & \\ \text{s.t.} & \sum\limits_{i=1}^{n} x_i = m & \\ & x_i \in \{0,1\} & i = 1, \ldots, n. \end{array} \tag{1}$$

This formulation is based on the binary variables x_i indicating whether object i is selected or not. Note that although this mathematical model only contains one objective function and one constraint, it is indeed quite complicated to solve, due to the combination of a non-linear objective function with a discrete solution space (as a result of the binary variables). On the other hand, the combinatorial nature of its solutions makes its complete enumeration impracticable, thus being a challenge for both exact and heuristic solving methods.

The second model in terms of its popularity is probably the *Max-Min Diversity Problem* (MMDP) (Erkut, 1990), in which the minimum distance between the selected elements is maximized. It can be formulated in a similar way as follows:

$$\begin{array}{lll} \text{(MMDP) Maximize} & z_{MM}(x) = \min\limits_{i<j} d_{i\,j} x_i x_j & \\ \text{s.t.} & \sum\limits_{i=1}^{n} x_i = m & \\ & x_i \in \{0,1\} & i = 1, \ldots, n. \end{array} \tag{2}$$

The Max-Sum and Max-Min literature includes extensive surveys (Ağca

Table 1: Diversity measures according to (Sandoya et al., 2018).

Measure	Mathematical function	Description
Sum	$\sum_{i<j,i,j\in M} d_{ij}$	This measure may address diversification among selected elements to distance.
Min	$\min_{i<j,i,j\in M} d_{ij}$	Focus on the minimum distance among the selected elements.
Mean	$\frac{\sum_{i<j,i,j\in M} d_{ij}}{\lvert M\rvert}$	Related to the Sum measure, is an average equity measure.
MinSum	$\min_{i\in M} \sum_{j\in M,j\neq i} d_{ij}$	This measure considers the minimum sum of distances, which corresponds to the aggregate dispersion among elements.
Diff	$\max_{i\in M} \sum_{j\in M,j\neq i} d_{ij} - \min_{i\in M} \sum_{j\in M,j\neq i} d_{ij}$	This measure can be understood as the difference between the largest and smallest values of the dispersion sum.

et al., 2000; Erkut and Neuman, 1989; Kuo et al., 1993), exact methods (Ağca et al., 2000; Ghosh, 1996; Pisinger, 2006), and heuristics (Ghosh, 1996; Hassin et al., 1997; Kincaid, 1992; Ravi et al., 1994; Resende et al., 2010). Although we can find other mathematical models to map the notion of diversity into functions and constraints, they did not receive much attention. Special mention deserves the work in (Prokopyev et al., 2009) in which four different models in the context of facility location and group selection are proposed. Table 1 summarizes the diversity measures known for a subset M of elements (Sandoya et al., 2018). Note that all of them have been proposed in the context of single-objective optimization.

In this paper we study the diversity maximization from a multi-objective point of view. In particular, we consider the Max-Sum and the Max-Min measures as objectives to be simultaneously maximized. We have called this problem the Bi-Objective Diversity Problem (BODP). The aim of BODP is to provide the user with different solutions (subsets of the given set) with different objective values. To tackle the BODP we have explored six different methods from three families of algorithms: the Non-dominated Sorting Genetic Algorithm-II, usually known as NSGA-II (Deb et al., 2002) and the second version of the Strength Pareto Evolutionary Algorithm, also known as SPEA2 (Zitzler et al., 2001), from the population-based algorithms; Greedy Randomized Adaptive Search Procedure (Feo and Resende, 1989) and Iterated Greedy (Ruiz and Stützle, 2007) from the construction-based algorithms; and Tabu Search (Glover and Laguna, 1997) and Variable Neighborhood Search (Hansen and Mladenovic, 2001) from the trajectory-based algorithms. We have adapted these six algorithms to the

BODP proposing some new elements, such us the codification of solutions, new genetic operators, constructive methods and local search strategies. Given that those algorithms require several parameters to be adjusted, we have used iRace (López-Ibánez et al., 2016) to automatically tune their configuration taking into account the hypervolume quality indicator. Finally, we have compared the six approaches in a well-known set of instances using three quality indicators for multi-objective optimization: hypervolume, set coverage, and epsilon indicator. The results show that the Tabu Search method obtains better results and spends shorter execution times.

The rest of the paper is organized as follows. In Section 2, we motivate the multi-objective nature of the problem. In Section 3, we describe the algorithmic proposals, while in Section 4, we detail the experimental experience. Finally, in Section 5, we draw the conclusions and the future work.

2. Problem motivation

Given that the main purpose of this paper is to maximize diversity, one may ask, why is it important? We can find numerous applications in the scientific literature to answer it. From a human resources perspective, promoting diversity and equity is a goal in many companies and organizations. This includes INFORMS, the American institute for operations research and management science. This institute even promotes an award, the WSC Diversity Award, to improve outreach and diversity among young researchers. It is nowadays a well-established principle in many companies that increasing human diversity in not only ethical but also beneficial for the company efficiency. This has been very-well summarized in the book by Scott Page (Page, 2007), where he states that "Diverse perspectives and tools enable collections of people to find more and better solutions and contribute to overall productivity".

If we turn our attention to logistics and consider for example the area of vehicle routing, we can find specific models to create diverse routes. Martí et al. (2009) considers a multi-objective routing problem in which a set of paths from an origin to a destination must be generated. Finding different paths in a graph is a classical optimization problem, and in the context of hazardous materials transportation, we want to obtain spatially dissimilar paths that minimize the risk (distributing the risk over all regional zones to be crossed uniformly). The Path Dissimilarity Problem involves obtaining a set of p paths with minimum length and maximum diversity. This and related models have been extensively studied in Operations Research where maximizing diversity is an important point.

In this paper we focus on the two most studied models, Max-Sum and Max-Min, and propose a bi-objective model for an improved approach to identify diverse subsets in a given set. We target these two objectives for several reasons. The first one is that they are pure diversity models, while the others mentioned above are equity models according to Sandoya et al. (2018). These authors reviewed all the existing models and classified them according to their scope. A second reason to consider these two objectives is because of their relatively low

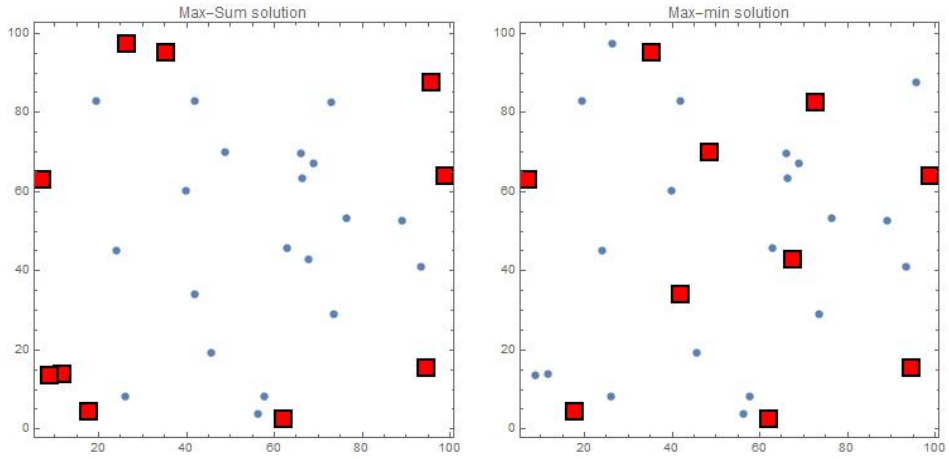

Figure 1: Comparison of optimal solutions for Max-Sum (left) and Max-Min (right) on the same instance.

correlation (Resende et al., 2010). Therefore, we should not expect a method designed for one of these problems to obtain good solutions for the other one. Finally, a third reason to select these objectives in our study is related to the composition of the solutions obtained by search methods. In particular, as described in Sandoya et al. (2018), the solutions obtained with one model are usually very different to the solutions obtained with the other model.

To illustrate this point, we have run the following experiment. Figure 1 shows the optimal solution of both models on a small example with $n = 30$ elements from which we want to select $m = 10$. This is a toy example where the axes of the figure indicate coordinate values, and the dots represent specific locations. Squares correspond to the selected elements. This example was generated from scratch as a proof of concept of our proposal. It is clear from the example in Figure 1 that, depending on the model considered, we can obtain solutions with a very different structure (in terms of the position of the selected elements). Although in the last few years many algorithms have been proposed to solve these two models, we are not aware of any analysis of the solutions obtained. It has not been studied either which model suits better for each type of application.

We can consider that the Max-Sum model is somehow more complete than the Max-Min model since it considers all the pairwise distances when computing the objective function. On the other hand, the Max-Min model forces the closest selected elements to be as far as possible, preventing similar (close) elements to appear in the solution. A potential user may argue that she or he would like to consider these two desirable characteristics in the solution. Therefore, an approach such as the BODP proposed here, satisfies this kind of requirements.

To illustrate the advantages of a bi-objective approach, we have performed a

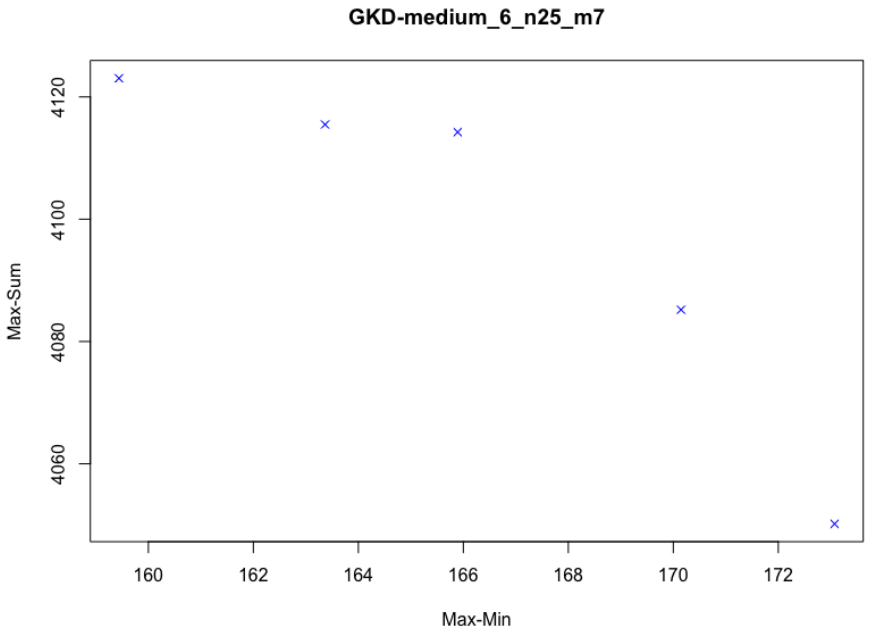

Figure 2: Exact Pareto front for a small instance with $n = 25$ and $m = 7$.

simple experiment. We consider a small instance (with $n = 25$) and generate all its feasible solutions (subsets of $m = 7$ elements), and compute for each solution the two objectives. Then, we only take non-dominated solutions (i.e., those for which there are no other solution with a better value in both objectives). With this enumeration we obtain the so-called exact Pareto front.

Figure 2, which displays the objective value of the solutions, shows that the Pareto front contains five solutions, being the one in each extreme of the optimal solution of each respective single optimization problem. The existence of three efficient solutions between the two extreme solutions confirms that the bi-objective model has an additional value with respect to the single objective models. In practical terms, the resolution of the bi-objective model can provide the user with several solutions potentially interesting to choose the one that fits better in the domain considered. In addition to the enumeration, we have conducted this experiment using the heuristic methods that we will present in the following section, and all of them were able to reach the optimal solution because this is a small instance.

Table 2: Classification of the studied methods.

Population-based	Construction-based	Trajectory-based
NSGA-II	GRASP	Tabu Search
SPEA2	Iterated Greedy	VNS

3. Algorithmic Proposals

Once we have presented the Bi-Objective Diversity problem and motivated the optimization of the two objectives simultaneously, we next describe the algorithmic proposals studied in this work. We have grouped our proposed methods according to the standard classification of population-based, constructive-based, and trajectory-based methods. Table 2 shows the methods under this classification. The following three subsections are respectively devoted to them.

3.1. Population-based methods

In the field of metaheuristics, there are many approaches that are based on maintaining a population of individuals, each one representing a solution of the given problem. Among them, we can find Genetic Algorithms, Genetic Programming, Scatter Search, Ant Colony Optimization, and algorithms related to different swarms of elements. In order to tackle our bi-objective problem, we have selected the classical NSGA-II and SPEA2 genetic algorithms to represent this class of methods. Our adaptation of both methods to solve the BODP follows.

Before going through the algorithmic details, we would like to briefly explain two common terms in the field of genetic algorithms that we will next use: chromosome and gene. A chromosome is the representation of a solution by means of a list (or array) of integer values. Each one of the components of a chromosome is a gene, and encodes a small number of features of the solution (typically one feature for each gene).

3.1.1. NSGA-II and SPEA2

According to recent surveys (see for example Sayyad and Ammar (2013)), the two most popular algorithms in multi-objective optimization are the second version of the NSGA-II (Deb et al., 2002) and the improved version of the Strength Pareto Evolutionary Algorithm, which is known as SPEA2 (Zitzler et al., 2001).

NSGA-II is considered the *de facto* standard when the number of objectives is small. According to Deb and Jain (2014), multi-objective optimization corresponds to a number of two or three objectives, which is a small number of them. In the case of four or more objectives, this optimization is known as many-objective optimization, and variations of the state-of-the-art algorithms

are proposed to deal with this number of objectives. This is the case of NSGA-III (Deb and Jain, 2014), for example. NSGA-II is used in more than 53% of the papers reviewed in Sayyad and Ammar (2013).

The pseudo-code of NSGA-II is shown in Algorithm 1. The execution begins with the generation of an initial random population P of Pop solutions formed by chromosomes. In this case, a chromosome is an array of integer values, called genes, that encode a solution. Each gene stores the identifier of a selected node. Hence, the length of the chromosomes is p. Then, the evaluation of the solutions is performed in step 2 and, after that, the main loop iterates during Gen generations. The crossover operator is applied to the population P using the probability $CxPr$ in step 5. The mutation operator is applied to the population generated after the crossover, P_{Cx} with probability $MtPr$ in step 6. The resulting population, P_{Mt} is joined with the previous populations in step 7, producing a new population P_{All}, which is evaluated in step 7. After the evaluation, the crowding distance of the members of P_{All} is computed in step 9. The crowding distance measures the density of solutions around a selected one. This metric is used in conjunction with the dominance in the ranking of solutions in step 10 and, using this ranking, the population is reduced to the original Pop size in step 11. The loop ends once the maximum number of generations is reached.

Algorithm 1: Pseudo-code of the NSGA-II algorithm.

1: $P \leftarrow$ `randomPopulation`(Pop)
2: `evaluate`(P)
3: $i \leftarrow 0$
4: **while** $i < Gen$ **do**
5: $\quad P_{Cx} \leftarrow$ `executeCrossover`$(P, CxPr)$
6: $\quad P_{Mt} \leftarrow$ `executeMutation`$(P_{Cx}, MtPr)$
7: $\quad P_{All} \leftarrow P \cup P_{Cx} \cup P_{Mt}$
8: $\quad$ `evaluate`(P_{All})
9: $\quad$ `computeCrowdingDistance`(P_{All})
10: $\quad$ `assignRank`(P_{All})
11: $\quad P \leftarrow$ `reducePopulationThroughCrowding`(P_{All}, Pop)
12: $\quad i \leftarrow i + 1$
13: **end while**
14: $ND \leftarrow$ `obtainNonDominated`(P)
15: **return** ND

The SPEA2 algorithm follows a similar general scheme, based on chromosomes and genetic operators. However, it differs from NSGA-II in the particular features that are taken into account for the individuals.

As seen in Algorithm 2, the steps before the main loop are exactly the same as in NSGA-II but step 4, where the archive of solutions A is initialized. The termination condition of the main loop is also the number of generations, but the internals of the loop are different. Inside the loop, step 6 joins the

current population with the archive in P_{All}, and the strength of these solutions is calculated in step 7. Then, a reduction of P_{All} is performed in step 8, where the strength and the cost of each solutions is taken into account for the removal of solutions. In this step, if the remaining number of solutions is lower than Pop, the population is completed with solutions from A. After this process, the crossover and mutation operators are applied, and the current population after them is evaluated, looping till the final generation. After the last iteration, the non-dominated solutions are returned.

Algorithm 2: Pseudo-code of the SPEA2 algorithm.

1: $P \leftarrow$ `randomPopulation`(Pop)
2: `evaluate`(P)
3: $i \leftarrow 0$
4: $A \leftarrow \emptyset$
5: **while** $i < Gen$ **do**
6: $\quad P_{All} \leftarrow P \cup A$
7: $\quad$ `calculateStrength`(P_{All})
8: $\quad P \leftarrow$ `reducePopulationThroughStrength`(P_{All}, A, Pop)
9: $\quad P_{Cx} \leftarrow$ `executeCrossover`$(P, CxPr)$
10: $\quad P_{Mt} \leftarrow$ `executeMutation`$(P_{Cx}, MtPr)$
11: $\quad P_{All} \leftarrow P \cup P_{Cx} \cup P_{Mt}$
12: $\quad$ `evaluate`(P_{All})
13: $\quad i \leftarrow i + 1$
14: **end while**
15: $ND \leftarrow$ `obtainNonDominated`(P)
16: **return** ND

3.1.2. Adaptation to BODP

As stated before, both NSGA-II and SPEA2 are based in a population where each individual is represented by a chromosome. Therefore, in order to deal with the BODP we have adapted the codification and the genetic operators in the same way for both algorithms.

Regarding solution representation, we have codified them as integer chromosomes of length m. Hence, the chromosome contains the elements selected by the solution which, in the initial population, are randomly generated.

Regarding the genetic operators, we have considered the classical tournament selection, single-point crossover and the uniform mutation for both algorithms (Eiben and Smith, 2003). However, the straightforward implementations of crossover, and mutation could lead to unfeasible solutions due to repetitions of elements in the offspring. Figure 3 shows an example of crossover where the first offspring is unfeasible due to the repetition of element 5.

To overcome the limitation illustrated in Figure 3, we developed an implementation that we call *combinatorial crossover*, with the objective of avoiding unfeasible solutions. Algorithm 3 shows the steps performed by the operator.

Figure 3: Usual single-point crossover. First offspring is not a feasible solution.

As seen, given two parent solutions, P_1 and P_2, the first step is to obtain the common elements, called *fixed genes*, $FixG$. These elements will belong to the two offspring O_1 and O_2, as seen in steps 2 and 3. Those elements that are not fixed genes are called *free genes*, and are collected in set $FreeG$ in step 4. The free genes are then randomly distributed on the offspring, as shown in the loop that begins in step 5 and ends in step 12. Notice that function `selectRandom` in steps 6 and 9 randomly selects one element from the given set of genes. This function will be used later in other algorithms with the same behavior.

Algorithm 3: Combinatorial crossover operator.

1: $FixG \leftarrow P_1 \cap P_2$
2: $O_1 \leftarrow FixG$
3: $O_2 \leftarrow FixG$
4: $FreeG \leftarrow P_1 \cup P_2 \setminus P_1 \cap P_2$
5: **while** $(FreeG \neq \emptyset)$ **do**
6: $\quad g_1 \leftarrow$ `selectRandom`$(FreeG)$
7: $\quad O_1 \leftarrow O_1 \cup \{g_1\}$
8: $\quad FreeG \leftarrow FreeG \setminus \{g_1\}$
9: $\quad g_2 \leftarrow$ `selectRandom`$(FreeG)$
10: $\quad O_2 \leftarrow O_2 \cup \{g_2\}$
11: $\quad FreeG \leftarrow FreeG \setminus \{g_2\}$
12: **end while**
13: **return** (O_1, O_2)

Figure 4 shows two examples of the combinatorial crossover. As seen, the offspring keeps common genetic information, while and the uncommon one is randomly distributed.

The mutation operator follows the same strategy of avoiding unfeasible solution. This *combinatorial mutation* works as described in Algorithm 4. First, the elements not belonging to the given solution S are obtained in set NS in step 1. Then, an element to be removed r is randomly selected from the solution, and an element to be added is randomly selected from NS in steps 2 and 3. Finally, the mutated solution S' is composed in step 4. Notice that, given the nature of the crossover and mutation operators, the position of the genes in the chromosome is not relevant. Therefore, step 4 represents the insertion as a

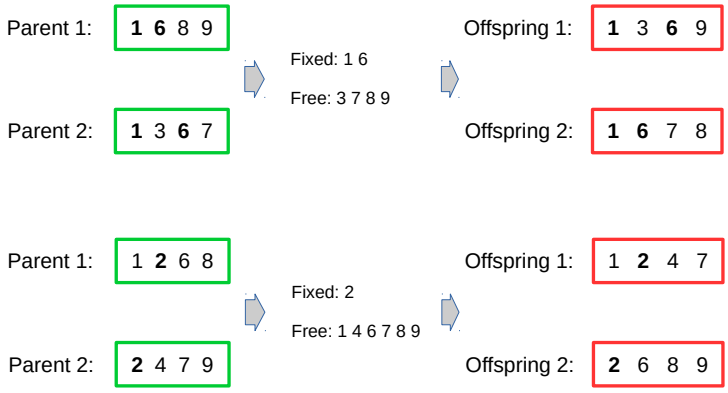

Figure 4: Example of combinatorial crossover.

union between a set and one element.

Algorithm 4: Combinatorial mutation operator.

1: $NS \leftarrow N \setminus S$
2: $r \leftarrow \texttt{selectRandom}(S)$
3: $a \leftarrow \texttt{selectRandom}(NS)$
4: $S' \leftarrow (S \setminus r) \cup a$
5: **return** S'

Therefore, these combinatorial operators maintain the algorithms in the space of feasible solutions without the need of any repair operation.

3.2. Construction-based methods

Instead of having a population of individuals, other optimization methods focus on the construction of solutions, which can be later modified in different ways. Therefore, the exploration of the solution space is different than in the case of population-based methods. In short, it is made by generation instead of by combination. Hence, in order to assess the performance of the construction-based methods, we have selected two methodologies that exhibit a remarkable performance when targeting mono-objective diversity optimization: Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Greedy. Both approaches create solutions using a constructive method based on a greedy function.

The proposed greedy function for this problem computes the objective value for a solution S once the selected element j is added. The input parameter obj determines which one of the cost functions (corresponding to both objectives) has to be taken into account. More formally, the greedy function $g(S, j, obj)$ is defined as follows:

$$g(S,j,obj) = \begin{cases} \min\limits_{i<k,i,k\in\{S\cup\{j\}\}} d_{ik} & \text{if } obj = 0 \quad (Max - Min) \\ \sum\limits_{i<k,i,k\in\{S\cup\{j\}\}} d_{ik} & \text{if } obj = 1 \quad (Max - Sum) \end{cases} \tag{3}$$

3.2.1. GRASP

The GRASP methodology, introduced in Feo and Resende (1989), performs a given number of iterations in which a trial solution is constructed and then improved, typically using a local search method. The GRASP method is greedy because the elements that are added to a trial solution in the construction phase are selected according to a greedy function, but the selection of the elements is randomized through a list of candidates and a selection parameter typically named α.

To tackle this problem, we propose a constructive method that alternatively considers each one of the objectives when constructing subsequent solutions. That is, once a solution is constructed according to an objective, this current objective is changed. The pseudo-code of this constructive procedure is shown in Algorithm 5. Firstly, the solution S is reset and the candidate list CL is filled in with all the elements of the instance, which are initially stored in J, as seen in steps 1 and 2, respectively. Then,the main loop begins by adding elements to S. To this aim, the minimum and maximum values for the current objective are obtained for all the candidate elements in steps 4 and 5. These values are used in the creation of the restricted candidate list RCL in step 6, which is guided by the parameter α. This parameter biases the selection of the elements according to their quality. If $\alpha = 0$, any element can be selected. If $\alpha = 1$, only the element with the highest objective value will be selected. In step 7 an element from the RCL is randomly selected and both, the current solution and CL, are updated in steps 8 and 9. The loop ends when the solution reaches the required size, which is m. Finally, once the solution is built, the objective is alternated with the modulus function, as shown in steps 11 to 15.

Given that this method is a greedy constructive that alternates objectives between different solutions, we call it C_{AltBwS}. In addition this method, and in line with Martí et al. (2015), we will use a different one that alternatively considers each one of the objectives when including an element in the solution. We call this method C_{AltInS}. Finally, in order to get baseline for our empirical comparisons, we have also tested a method which randomly creates a solution. This procedure is called C_{Rnd}. Therefore, we propose three different constructive methods whose performance will be analyzed in the experimental experience.

After the execution of a constructive method, a local search process is performed on the incumbent solution in the GRASP algorithm. The design of the local search for a multi-objective problem has taken a lot of attention in the literature, as highlighted in a recent PhD thesis (Dubois-Lacoste, 2014). In that work, we may find two main approaches: the scalarization of the objectives through a weighted-sum function, and the use of dominance in the search process.

Algorithm 5: Greedy constructive algorithm (C_{AltBwS}).

$S \leftarrow \emptyset$
CL $\leftarrow J$
while $|S| < m$ **do**
  $g_{min} \leftarrow \min_{j \in \text{CL}} g(S, j, obj)$
  $g_{max} \leftarrow \max_{j \in \text{CL}} g(S, j, obj)$
  RCL $\leftarrow \{j \in \text{CL} \mid g(S, j) \geq g_{min} + \alpha \cdot (g_{max} - g_{min})\}$
  $j \leftarrow$ `selectRandom`(RCL)
  $S \leftarrow S \cup \{j\}$
  CL $\leftarrow$ CL $\setminus \{j\}$
end while
if $obj = 0$ **then**
  $obj \leftarrow 1$
else
  $obj \leftarrow 0$
end if
return S

We have tried the scalarization approach in preliminary experiments on the BODP problem, but our results were not very competitive. In fact, the local search was focused on the convex part of the Pareto front, but the density of solutions in that area was so low that the local search process was not able to produce a significant number of non-dominated solutions.

Therefore, we implemented three local search algorithms with different strategies, all of them following the first improvement scheme. In addition, the three algorithms explore neighbor solutions with the same exchange operator which, given a solution S, an element $i_1 \in S$ and an element $i_2 \notin S$, it produces a new solution S' such that $S' = S \setminus \{i_1\} \cup \{i_2\}$. Thus, the neighborhood is formed by all the solutions which are different from the incumbent solution in one element.

The first local search algorithm, LS_{Dom} is based on dominance. The method explores the neighborhood of a given solution as defined above. An improvement in the incumbent solution is produced if and only if a dominant solution is found. This strategy ends when none of the solutions in the neighborhood dominates the current one. Notice that in the search process, non-dominated solutions will be found. These solutions are stored in an archive and, at the end of the process, will be examined to obtain the final set of non-dominated solutions.

The second local search follows a different approach. In particular, it alternates the improvement by considering either Max-Min or Max-Sum objectives in subsequent solutions. That is, if Max-Min was the objective to improve in a given solution, the next solution will be improved on the Max-Sum objective, and the other way around. We call this local search LS_{Alt}. Given that this is a bi-objective problem, the traversed solutions are stored in an archive and, as in the previous local search, they are examined at the end of the search to obtain

the non-dominated solutions.

The third local search takes a solution and improves it using both the Max-Sum and the Max-Min objectives by applying a path relinking (PR) process. We call this local search LS_{PR}. PR was proposed as a way of including intensification and diversification in the context of tabu search (Glover, 1996; Glover and Laguna, 1997), and later was included as an intensification method in the GRASP methodology (Laguna and Martí, 1999). In our local search proposal, PR generates a path from the initial solution (S_1 in our case) towards the guiding solution (S_2). Figure 5 shows an example considering four elements on each solution. As seen in the figure, four solutions are generated from S_1 at exchanging each element with one of the guiding solution S_2. In our proposal, the next element in the path is selected by random. In the example, the selected solutions are highlighted with different background. The path ends once the incumbent solution is changed to produce the guiding solution.

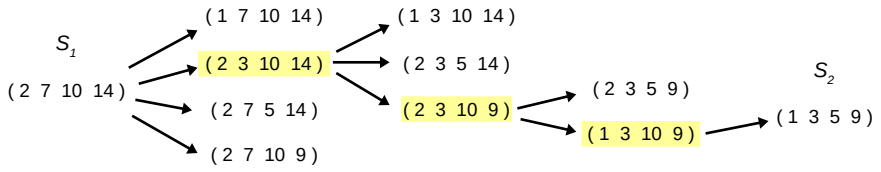

Figure 5: Example of Path Relinking from S_1 to S_2.

The pseudo-code of LS_{PR} is shown in Algorithm 6. As seen, two improved solutions S_1 and S_2 are obtained as a result of maximizing Max-Sum and Max-Min objectives respectively starting from the same incumbent solution. Then, in step 3, we apply PR to traverse the space of solutions from S_1 to S_2. As occurs in the case of the local search methods already described, the non-dominated solutions that are traversed are stored in an archive called A in the pseudo-code. This archive is also used by both `localSearchMaximizeMin` and `localSearchMaximizeSum` in order to try to reach as many non-dominated solutions as possible. Finally, the non-dominated solutions from A are obtained in step 4.

Algorithm 6: Path relinking local search, LS_{PR}.

1: $S_1 \leftarrow$ `localSearchMaximizeMin`(S)
2: $S_2 \leftarrow$ `localSearchMaximizeSum`(S)
3: $A \leftarrow A \cup$ `pathRelinking`(S_1,S_2)
4: $ND \leftarrow$ `obtainNonDominated`(A)
5: **return** ND

As a result of combining these elements, the steps that form the GRASP proposals are shown in Algorithm 7. It receives four input parameters: $CrAlg$,

which is the identifier of the constructive algorithm to be run; α, which is used by the constructive algorithm; and $maxConstr$, which states the maximum number of constructions to be built; and LS, which is the identifier of the local search method to be run. As seen in the code, the first steps are devoted to initialize the archive of solutions and to reset the number of constructions (steps 1 and 2). Then, the main loop begins with the greedy constructive method producing a solution S in step 4 using the corresponding constructive method. This solution is given as the starting solution for the local search method which, at it will be shown in the experimentation, can be any of the three procedures presented above, and will be determined before running the algorithm. This method returns the set of solutions that were traversed, updating the archive A in step 5. The number of constructions is then incremented, and the loop iterates until the number of constructions reaches the given maximum value. Finally, the non-dominated solutions from A are obtained in step 8.

Algorithm 7: Proposed multi-objective GRASP method scheme.

1: $A \leftarrow \emptyset$
2: $constr \leftarrow 0$
3: **while** $constr < maxConstr$ **do**
4: $\quad S \leftarrow$ `constructiveMethod`$(CrAlg, \alpha)$
5: $\quad A \leftarrow A \cup$ `localSearch`(S, LS)
6: $\quad constr \leftarrow constr + 1$
7: **end while**
8: $ND \leftarrow$ `obtainNonDominated`(A)
9: **return** ND

3.2.2. Iterated Greedy

The Iterated Greedy (IG) algorithm (Ruiz and Stützle, 2007) generates a candidate solution using a constructive method. The incumbent solution is partially destroyed and then is re-built using again a constructive method. This process is repeated until a stopping condition is met. Due to the simplicity of the method, it is usually fast and easy to implement it.

In Algorithm 8 we show the pseudo-code of the proposed IG algorithm. As seen, the first statement generates a solution S with the constructive method indicated by $CrAlg$. Then, this solution is stored in an archive, A, which will keep all the solutions traversed by the algorithm. We account for the number of partial destructions of solutions using the variable $numDestr$. This way, the stopping condition requires $numDestr$ to be lower than the maximum number of destructions, indicated by the input parameter $maxDestr$, as seen in step 4. In step 5, solution S is partially destroyed, according to a percentage of destruction given by the input parameter pct, obtaining the partial solution S'. The selection of the elements to be removed from the solution is always at random. Next, this solution is rebuilt using the same method as in step 1. The resulting solution, S'' is added to the archive and the number of destructions is

incremented, as seen in steps 7 and 8. Once the stopping condition is met, the non-dominated solutions are obtained in step 10.

Algorithm 8: IG method.

1: $S \leftarrow$ `constructiveMethod`$(CrAlg, \alpha)$
2: $A \leftarrow \{S\}$
3: $numDestr \leftarrow 0$
4: **while** $numDestr < maxDestr$ **do**
5: $\quad S' \leftarrow$ `partiallyDestroy`(S,pct)
6: $\quad S'' \leftarrow$ `rebuild`(S')
7: $\quad A \leftarrow A \cup \{S''\}$
8: $\quad numDestr \leftarrow numDestr + 1$
9: **end while**
10: $ND \leftarrow$ `obtainNonDominated`(A)
11: **return** ND

3.3. Trajectory-based methods

In the trajectory-based optimization methods, the common feature is the generation of a number of consecutive solutions that form a path which, ideally, reaches an optimal solution. The selection criterion to reach the next solution is not always guided by an improvement strategy. On the other hand, moves to worse solutions (in terms of quality) are allowed to avoid local optima stagnation. Among these methods, we have selected Tabu Search (TS) and Variable Neighborhood Search (VNS) as representative ones to be applied to this problem.

For these two approaches we have followed a similar idea, which was presented in Duarte et al. (2015) for trajectory-based optimization methods. The authors propose to consider the current set of non-dominated solutions, namely approximation to the Pareto front, as the incumbent solution. Hence, if the front changes after exploring a neighborhood, it means that new non-dominated solutions have been added to the front and, therefore, an improvement has been made. On the other hand, if the exploration has not found new non-dominated solutions, the Pareto approximation set will not be modified, meaning that no improvement has been obtained.

3.3.1. Tabu Search

Tabu search (Glover and Laguna, 1997) is a trajectory-based algorithm which explores the neighborhood of the current solution taking the best solution in the neighborhood as the next element in the trajectory. The best solution in the neighborhood may not be better than the current one. However, paths that decrease the quality of the current solution may lead to local optima, and over the long term to global optimal solutions (in single-objective optimization). In the classical implementation of TS, some of the characteristics, called attributes, of the traversed solutions, are stored in the tabu list with the aim of avoiding

them in subsequent explorations. However, in the BODP we have decided to store the solution in tabu list. More precisely, we store the hash code that corresponds to each traversed solution. Actually, instead of storing the whole solution, we compute a hash value (with a standard function) and include that value in a hash map. This strategy allows us to determine if a solution is tabu-active in $\mathcal{O}(1)$.

In Algorithm 9 we show our TS proposal. The first three steps are devoted to initialize the tabu list (TB), the set of non-dominated solutions (ND) and the counter of the iterations where no improvement was made ($itersNoImprove$). With this later element we establish the termination condition, which is defined by the maximum number of iterations performed with no improvement, set by the input parameter $maxItNoImpr$. In step 4, a solution S is built with the corresponding constructive method, and then the main loop begins. In step 6, the current number of non-dominated solutions is stored into variable n_{ND}. After that, the exploration of the neighborhood of S is performed. Notice that the `exploreNeighbohood` method receives S as the current solution, LS as the identifier of the local search method to be run, ND which can be updated, T as the tenure parameter for the tabu structure, and TB which will be also updated by the exploration. In step 8, the new size of ND is compared to the previous stored n_{ND}. In the case they are equal, no improvement is made because we consider that two fronts with the same number of non-dominated solutions are equivalent. Then, a solution which is not tabu is randomly selected from ND, its hash code is included in the tabu list and the number of iterations with no improvements is increased (steps 9 to 11). Finally, the set of non-dominated solutions is returned.

Algorithm 9: TS method.

1: $TB \leftarrow \emptyset$
2: $ND \leftarrow \emptyset$
3: $itersNoImprove = 0$
4: $S \leftarrow$ `constructiveMethod`$(CrAlg, \alpha)$
5: **while** $itersNoImprove < maxItNoImpr$ **do**
6: $\quad n_{ND} \leftarrow |ND|$
7: $\quad ND \leftarrow$ `exploreNeighbohood`(S, LS, ND, T, TB)
8: $\quad$ **if** $|ND| = n_{ND}$ **then**
9: $\quad\quad S \leftarrow$ `randomSelection`$(ND \setminus TB)$
10: $\quad\quad TB \leftarrow TB \cup \{$ `hash(`S`)` $\}$
11: $\quad\quad itersNoImprove \leftarrow itersNoImprove + 1$
12: $\quad$ **end if**
13: **end while**
14: **return** ND

The constructive method that is called in step 4 can be any of the ones described in Section 3.2.1. In addition, the `exploreNeighbohood` method implements the three local search procedures described in Section 3.2.1, but taking

into account that the moves of the local search have to skip the elements in the tabu list. This way, we will be able to measure the impact of the algorithms using similar constructive and improvement strategies.

3.3.2. Variable Neighborhood Search

The Variable Neighborhood Search metaheuristic is a trajectory-based method basically formed by two steps: a perturbation phase and a descendant phase. In the first one, which is an exploration step, it takes into account changes of neighborhood to get out of a basin of attraction. In the second one, it performs an intensification of the search to find a local optimum by means of a local search (Hansen and Mladenovic, 2001). As in the case of TS, it creates a trajectory of solutions which not always increase the quality of the current solution locally, but performs an intelligent exploration of the solution space, finding very high-quality solutions, if not the global ones.

Algorithm 10: VNS method.

```
1:  ND ← ∅
2:  i ← 0
3:  S ← constructiveMethod(CrAlg, α)
4:  while i < maxIters do
5:    k ← 1
6:    while k ≤ k_max do
7:      S' ← shake(S, k)
8:      size_ND ← |ND|
9:      ND ← improvementMethod(S', LS, ND)
10:     k ← k + 1
11:     if |ND| ≠ size_ND then
12:       k ← 1
13:       S' ← randomSelection(ND)
14:     end if
15:   end while
16:   i ← i + 1
17: end while
18: return ND
```

Several variations of VNS can be found in the literature. However, we consider the Basic VNS scheme in this paper, as shown in Algorithm 10. Instead of the classical termination condition guided by the execution time, we have determined that the main loop will run a number of iterations, stated by the input parameter $maxIters$. After the initialization, a solution is built in step 3 with the corresponding constructive algorithm. Then, the main loop begins, where the perturbation parameter k is reset in step 5. Next, the secondary loop begins, which is devoted to maintain k under the maximum perturbation size given by the input parameter k_{max}. Then, the shaking mechanism is executed in step 7, perturbing the incumbent solution by exchanging k elements,

randomly selected from the solution, with other k elements, randomly selected among those not included in S.

Then, the shaking mechanism is executed in step 7, perturbing the incumbent solution by exchanging k elements randomly selected from the solution with k elements randomly selected among those not included in S. Then, as in the TS scheme, the size of the non-dominated set of solutions is obtained in step 8. After that, the improvement method LS is run on the current neighborhood and k is incremented in steps 9 and 10. If the size of the non-dominated set has changed, it means that an improvement has been reached. Therefore, k is set to 1 in step 12, and the current solution is updated by randomly select a solution from the non-dominated set in step 13. If the maximum perturbation is reached, then the number of iterations is increased in step 16, and the main loop iterates. Finally, the set of non-dominated solutions is returned. As in the case of TS, we will configure VNS with the same constructive and improvement methods than those described in Section 3.2.1.

4. Experimental results

We perform our empirical comparison of the methods described in the previous sections on a benchmark instances from the MDPLIB, publicly available at `http://www.optsicom.es/mdp/`. In particular, we focus on the **GKD** data set, which comprises a number of 145 instances. The set is divided into three subsets:

- **GKD-small**: these 75 instances were introduced in (Glover et al., 1998). Here, the number of coordinates for each point is generated randomly in the 2 to 21 range, where $2 \leq n \leq 21$ represents a n-dimensional space. The instance sizes are such that for n = 10, m = 2, 3, 4, 6 and 8; for n = 15, m = 3, 4, 6, 9 and 12; and for n = 30, m = 6, 9, 12, 18 and 24.
- **GKD-medium**: these 50 matrices were introduced in (Martí et al., 2010). The number of coordinates for each point is generated randomly in the 2 to 21 range and the instance sizes are such that for n = 25, m = 2 and 7; for n = 50, m = 5 and 15; for n = 100, m = 10 and 30; for n = 125, m = 12 and 37; and for n = 150, m = 15 and 45.
- **GKD-large**: these 20 matrices were introduced in (Duarte and Martí, 2007). They were generated with 10 coordinates for each point and n = 500 and m = 50.

The algorithms were coded in Java, and the experiments where run on a personal computer provided with an Intel i7 @ 2.9 GHz CPU and 8 Gb of RAM using OSX 10.12.5 and JDK 7.

4.1. Iterated race tuning

The algorithms described in Section 3 require several input parameters. Some of them have already been mentioned in the description of the methods, and the others are described and set in this section. The parameters of

the algorithms determine both the computation effort and the execution time. Therefore, we consider that a comparison among the best configuration of each algorithm will allow us to fairly compare their performance on this problem. Besides, we deal with a relatively large number of parameters and methods.

In order to find the most appropriate settings for the proposed method, we have used iRace (López-Ibánez et al., 2016). This software implements an iterated race procedure able to find the configuration that obtains the best results. In other words, its goal is to automatically configure optimization algorithms by finding the most appropriate settings given a set of instances of an optimization problem. Considering that each configurable parameter has associated a sampling distribution, iRace iteratively executes race procedures in three phases:

1. A race starts with a finite set of candidate configurations.
2. At each step of this phase, candidate configurations are evaluated on a single instance. After a number of steps, those candidate configurations that perform statistically worse than at least another one are discarded, and the race continues with the remaining surviving configurations. This procedure continues until reaching a minimum number of surviving configurations, or a maximum number of instances that have been used, or a number of executed experiments per race.
3. The update of the distributions consists in modifying the mean and the standard deviation in the case of the normal distribution, or the discrete probability values of the discrete distributions.

Once a race is finished, it starts the next race. Specifically, new configurations are generated after sampling probability distributions from the previous race. As well as a set with the union of the new configurations and the elite configurations from the previous race is generated. The algorithm finally stops when it reaches the total number of experiments or the number of candidate configurations to be evaluated at the start of a race is not greater than the number best configurations.

In our case, we have configured iRace to maximize the hypervolume of the front obtained as a solution. This indicator measures the volume that is covered by a given set of non-dominated solutions in the space of objectives (Zitzler et al., 2003). In the particular case of a two-objective problem, this indicator evaluates the area under an approximation of the Pareto front. Note that since we are maximizing the value of the objectives in BODP, the higher the hypervolume, the better the result. Besides, the hypervolume is a metric that does not require a reference set. Therefore, it can be calculated for each independent execution of the algorithm.

For the first set of experiments, we have selected 15 instances from **GKD-small**, 10 from **GKD-medium** and 4 from **GKD-large**, which form a total number of 29 instances, corresponding to the 20% of the complete set of instances. The selected instances present different sizes and features, with the aim of being representative of the **GKD** data set. They constitute our "training set" to configure the algorithms and to avoid the over-training when perform the final comparison over the whole set of instances.

In Table 3, we show the parameters and ranges of values for the population-based algorithms. Both NSGA-II and SPEA2 share the same number and type of parameters: number of generations, size of the population, probability of mutation and probability of crossover. As seen in the table, we have selected typical ranges of integer numbers for both, the generations and the population parameters, according to Eiben and Smith (2003). In the case of probability parameters, we let iRace to select a real value between 0.01 and 0.99, with a precision of 2 decimal figures. Notice that for this table and the following ones, we show in parentheses the abbreviation of each one of the parameters, which will be used in Table 6, that shows the results of the iRace execution.

Table 3: Parameters and ranges tuned by iRace for the population-based algorithms.

Parameter	**Range**	**Algorithm**
Generations (*Gen*)	[100, 50000]	NSGA-II & SPEA2
Population (*Pop*)	[50, 500]	NSGA-II & SPEA2
Mutation Prob. (*MtPr*)	[0.01, 0.99]	NSGA-II & SPEA2
Crossover Prob. (*CxPr*)	[0.01, 0.99]	NSGA-II & SPEA2

The parameters of the construction-based algorithms are shown in Table 4. Specifically, GRASP and IG share the parameter α and the constructive algorithm, whose values correspond to the methods described in Section 3.2.1: alternate objective in the same solution (C_{AltInS}), alternate objective between different solutions (C_{AltBwS}), and random constructive (C_{Rnd}). In the case of GRASP, iRace has to determine the number of constructions to be made, as well as the local search method to apply. The local search methods, also described in Section 3.2.1, are based on: dominance (LS_{Dom}), alternate objectives (LS_{Alt}), and both objectives with path relinking (LS_{PR}). In the case of the IG algorithm, iRace has to set the number of destructions and the percentage of destruction of the solutions.

Table 4: Parameters and ranges tuned by iRace for the construction-based algorithms.

Parameter	**Range**	**Algorithm**
Num. Constr. (*maxConstr*)	[10, 500]	GRASP
Local Search (*LS*)	LS_{Dom}, LS_{Alt}, LS_{PR}	GRASP
Num. Destr. (*maxDestr*)	[10, 10000]	IG
Pct. Destr. (*pct*)	[0.00, 100]	IG
α	[0.00, 1.00]	GRASP & IG
Constr. Alg. (*CrAlg*)	C_{AltInS}, C_{AltBwS}, C_{Rnd}	GRASP & IG

The parameter of trajectory-based algorithms are shown in Table 5. Specifically, TS and VNS share the α parameter, the constructive algorithm and the

local search method. The possible values for the last two parameters are the same than in the case of the construction-based algorithms, as already explained in Section 3.3. TS requires a decision on the number of iterations with no improvement, as well as on the *tenure* parameter. The tenure regulates the size of the tabu list. Given that the instances present different number of elements, we configured the tenure as a percentage. Hence, the range is an integer value between 1 and 100. For the VNS algorithm, the maximum number of iterations has to be also determined, in addition to the maximum value of the perturbation parameter, k_{max}. Again, given the different size of the instances, we have defined k_{max} as a percentage of the size of the solution. However, we have narrowed the range of values from 10.00 to 75.00 to reduce the search space in this case.

Table 5: Parameters and ranges tuned by iRace for the trajectory-based algorithms.

Parameter	**Range**	**Algorithm**
Iters. no Improve ($maxItNoImpr$)	[1,100]	TS
Tenure (T)	[1, 100]	TS
Max. Iterations ($maxIters$)	[1, 10]	VNS
k_{max}	[10.00, 75.00]	VNS
α	[0.00, 1.00]	TS & VNS
Constr. Alg. ($CrAlg$)	C_{AltInS}, C_{AltBwS}, C_{Rnd}	TS & VNS
Local Search (LS)	LS_{Dom}, LS_{Alt}, LS_{PR}	TS & VNS

We have configured iRace to run a maximum of 500 experiments for each algorithm using a precision of 2 decimal positions and the default confidence level. We have selected 500 experiments because in a preliminary experimentation, we configured iRace with higher number of experiments and the results were similar. The resulting parameters are shown in Table 6. We then configure each multi-objective procedure with the parameters found with iRace.

4.2. Comparison of the proposed algorithms

In the comparison among the proposed algorithms, we consider each one of the 145 instances from the **GKD** data set by using the best configuration provided by iRace. It is worth mentioning that the comparison of multi-objective optimization procedures is a really complicated task since we compare sets of points instead of single solutions. In this paper, we follow the guidelines described in Knowles et al. (2006). Specifically, these authors propose the following evaluators as the most discriminant indicators to compare multi-objective methods: the hypervolume (Hv), the set coverage (SC), and the epsilon indicator (I_ϵ).

Table 6: Results of the iRace optimization.

NSGA-II		**SPEA2**		**GRASP**	
Gen	28130	*Gen*	12805	*maxConstr*	421
Pop	340	*Pop*	340	α	0.79
MtPr	0.08	*MtPr*	0.14	*CrAlg*	C_{AltBwS}
CxPr	0.26	*CxPr*	0.15	*LS*	LS_{Alt}
IG		**TS**		**VNS**	
maxDestr	5212	*maxItNoImpr*	70	*maxIters*	7
pct	55.39	*T*	64	k_{max}	29.49
α	0.95	α	0.94	α	0.19
CrAlg	C_{AltBwS}	*CrAlg*	C_{AltBwS}	*CrAlg*	C_{AltInS}
		LS	LS_{Alt}	*LS*	LS_{PR}

4.2.1. Hypervolume

As stated in Section 4.1, Hv is a measure of the area enclosed between the set of non-dominated solutions and the origin in a Cartesian space formed the objective functions. In this experiment, we run 10 times each algorithm on each one of the 145 instances. We then take the set of non-dominated solutions considering the 10 fronts for a given algorithm and instance, calculating the associated Hv of the corresponding front.

In Table 7, we show the results of the average Hv over the entire testing set of 145 instances. As seen, the population-based algorithms obtain the best results for the small instance set, **GKD-small**, with SPEA2 reaching the best Hv. On the contrary, the trajectory-based algorithms get the worse values. In the case of the medium-size instances, **GKD-medium**, there is no category leading the results, being the GRASP algorithm the one with the best average. Regarding the largest instances, **GKD-large**, the clear winner is TS, which obtains the best average Hv. Considering the average over all the instances, shown in the last row of the table, the best algorithm is GRASP, followed by TS. Both the GRASP and the TS algorithms perform a more efficient intensification around the non-dominated solutions, leading to better Hv results in the medium and large instances.

Table 7: Average Hv values for each set of instances.

Instance Set	NSGA-II	SPEA2	GRASP	IG	TS	VNS
GKD-small	775859.44	**775864.61**	775785.59	767180.38	774822.18	775177.37
GKD-medium	4595307.71	4548788.3	**4622978.94**	4423887.18	4603803.72	4584286.67
GKD-large	197542.72	182106.76	202046.05	180864.92	**203050.29**	199367.02
Average	1856236.62	1835586.56	**1866936.86**	1790644.16	1860558.73	1852943.69

To complement this experiment, we have also calculated the number of instances where a specific algorithm obtained the best Hv value among the six methods. We show in Table 8 these values. Numbers in brackets indicate the

amount of instances in each subset. As seen in the table, these results are similar to the averaged Hv. .

Table 8: Number of best Hv results for each set of instances.

Instance Set	NSGA-II	SPEA2	GRASP	IG	TS	VNS
GKD-small (75)	69	**71**	53	38	65	62
GKD-medium (50)	26	14	**31**	5	27	18
GKD-large (20)	0	0	3	0	**17**	0
Total (145)	95	85	87	43	**109**	80

In order to take a reference for the results, we have obtained, for each one of the instances, the set of non-dominated solutions across the executions of all the algorithms as a reference for comparisons. This set is an approximation of the actual *Pareto front*. We show in Figure 6, the distribution of the Hv values for the three sets of instances and the six algorithms. These results are normalized with respect to the Hv of the reference front obtained for each instance.

As seen in the figure, all algorithms perform similarly for **GKD-small** with the exception of IG, which presents a worse distribution. In the case of **GKD-medium**, IG is again the worst method. In addition, we also can appreciate that GRASP performs better than the other procedures. For **GKD-large** the best distribution is obtained with TS, followed by GRASP. These results are in line with the previous experiments. Therefore, we can state that, on average, our TS proposal is the best algorithm in terms of Hv for the instances considered.

4.2.2. Set coverage

The set coverage indicator is a metric that returns the relative coverage comparison of two sets of solutions (Zitzler et al., 2000). Given a set of non-dominated solutions ND and a reference set of solutions R, the $SC(ND, R)$ value is computed by using the following equation:

$$SC(ND, R) = \frac{|\{x' \in R; \exists x \in ND : x \succeq x'\}|}{|R|}$$

That is, if all the solutions in ND dominate (or are equal) to all solutions in R, $SC(ND, R) = 1$. Notice that the weakly domination of x over x', represented as $x \succeq x'$, means that x is better than or equal to x' in all the objectives. Symmetrically, $SC(ND, R) = 0$ means the opposite situation.

Hence, in order to compare the performance of each algorithm, we consider as the reference set R, the one constructed with the set of non-dominated solutions when considering 10 independent executions of each compared method (i.e., set of non-dominated solutions obtained in the experiment). Notice that in this metric, the higher the value, the better the algorithm.

In Table 9 we show the set coverage between the front obtained by executing 10 independent times each algorithm and the reference R. These results are

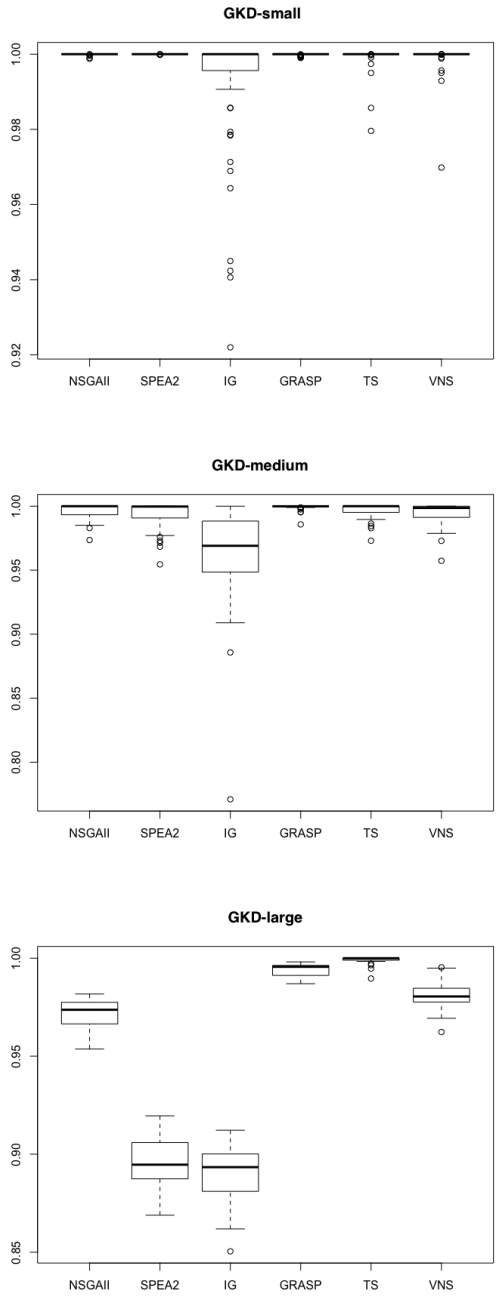

Figure 6: Boxplots showing the Hv distribution normalized with respect to the Hv of the Pareto front.

Table 9: Average set coverage measure $SC(ND, R)$, where ND is the set of non-dominated solutions obtained with each one of the algorithms under study.

Instance Set	NSGA-II	SPEA2	GRASP	IG	TS	VNS
GKD-small	**0.9729**	0.707	0.4122	0.2704	0.2467	0.2532
GKD-medium	**0.9093**	0.334	0.205	0.1	0.1366	0.1412
GKD-large	**0.5135**	0	0.0379	0	0.4491	0.0043
Average	**0.7985**	0.347	0.2184	0.1235	0.2775	0.1329

presented for each subset of instances and also averaged over the whole set of 145 instances. As it can be seen in the table, the best results are obtained by the NSGA-II algorithm. Besides, we can see that the population-based algorithms perform really well in **GKD-small** and **GKD-medium** data sets. In the case of **GKD-large**, the second best result is obtained by TS.

Analyzing these results, we can observe that the best performer in terms of Hv, TS, is ranked in the third position when considering set coverage. This result is due to the shape of the fronts. Specifically, TS obtains very good solutions in the extreme parts of the front, returning high hypervolume values. On the other hand, the results of NSGA-II are more concentrated in the convex part of the front (mainly due to the crowding distance operator). This way, it obtains a higher number of non-dominated solutions in relation to the reference front, while the values of Hv are not the best in several cases.

To illustrate this fact, we show in Figure 7 a comparison between the reference set (approximate Pareto front) and the non-dominated solutions obtained with NSGA-II and TS for the medium-size instance denoted as `GKD-medium_48_n150_m45`. In this case, the set coverage value of NSGA-II is 0.9285, while the TS method obtains 0.0714. Regarding the Hv, the value obtained with NSGA-II is 11200182.90, which is lower than the value obtained with TS, which is 11240882.56. As seen in the figure, TS obtains solutions in the lower-right part of the plot, which provide higher hypervolume. However, several of this points are dominated by the reference front. On the other hand, almost all the solutions found by NSGA-II match those solutions belonging to the reference front and, given that they do not reach the extreme part of the Max-Min objective, the hypervolume value is lower. Notice that this behavior have been observed in a considerable number of instances.

4.2.3. Epsilon indicator

Given two sets of non-dominated solutions, ND_1 and ND_2, the epsilon indicator, $I_\epsilon(ND_1, ND_2)$, corresponds to the smallest ϵ value which will move the front ND_1 in such a way that any solution belonging to ND_2 will be dominated by ND_1. We refer the reader to Zitzler et al. (2003) for a more detailed description. Hence, following the same idea as in the previous section, if we compute $I_\epsilon(ND, R)$, where ND is the set of non-dominated solutions obtained with a

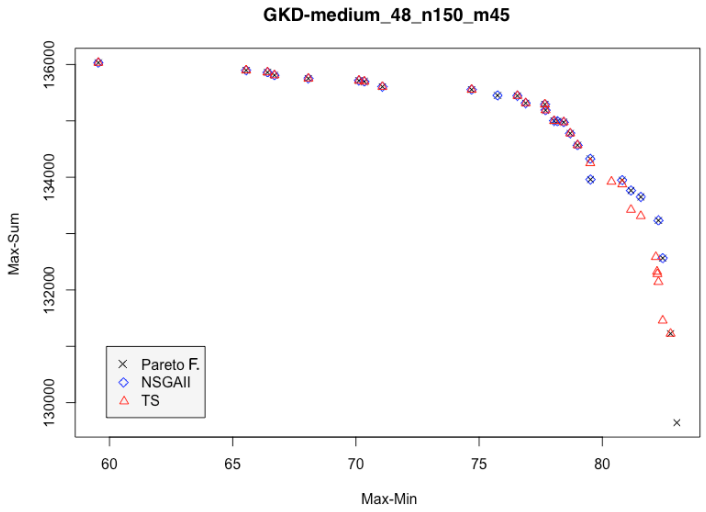

Figure 7: NSGA-II and TS fronts compared with the Pareto front (*Pareto F.*) in a medium-size instance.

Table 10: Average $I_\epsilon(ND, R)$ where ND is the set of non-dominated solutions obtained with each one of the algorithms under study.

Instance Set	NSGA-II	SPEA2	GRASP	IG	TS	VNS
GKD-small	**1.2126E-13**	0.0222	0.8757	28.6359	2.7728	12.9369
GKD-medium	18.5898	10.0240	43.2839	346.7773	**7.4116**	217.1936
GKD-large	12.4373	235.0064	121.8625	555.9344	**0.34946**	251.9661
Average	10.3424	81.6842	55.3407	310.4492	**3.5113**	160.6989

particular procedure when it is executed 10 times over each instance, and R is the approximation of the Pareto front obtained by merging all non-dominated solutions found by all algorithms. Then, the higher the value, the worse the front generated by the specific algorithm. We show the corresponding results in Table 10 where, again, the TS algorithm obtains the best overall results, despite it is not the best performer for the **GKD-small** set. Besides, the results of the NSGA-II are also good, obtaining the second best result in the overall average value.

4.2.4. Computing time

The execution time in this experimentation mainly depends on the configuration parameters of the algorithms. In our proposal, the configuration of each procedure was automatically obtained with iRace by taking into account the quality of the results given by the hypervolume metric. In other words, the

Table 11: Average execution time (in seconds) for each algorithm.

Instance Set	NSGA-II	SPEA2	GRASP	IG	TS	VNS
GKD-small	86.8121	969.8038	0.0233	0.0558	**0.0028**	0.0176
GKD-medium	178.805	800.5428	381.3414	1.8498	**0.365**	9.7940
GKD-large	626.7021	777.9501	149.6959	60.6636	**19.0266**	523.2950
Average	297.4397	849.4322	50.9546	20.8564	**6.4648**	177.7022

execution time is not considered when we tune each procedure. Nonetheless, we think that it is illustrative to give the information about the time spent by each algorithm run on each one of the instances. Those results are presented in Table 11 in seconds.

As seen in the table, TS is the fastest algorithm, followed by IG. Therefore, taking into account the methods and strategies proposed in this paper to deal with this BODP through different algorithms, TS appears to be the best choice. Not only because of the quality of the solutions, which has been analyzed with the three metrics described before, but also because it is the faster method among the six proposed algorithms. The NSGA-II algorithm also obtains good quality results, as explained in previous sections. However, it spends more execution time than TS.

5. Conclusions and Future Work

We propose in this paper a novel approach to the diversity problem. This approach considers the simultaneous maximization of two non-related objectives, which are the sum of the distances between the selected elements (Max-Sum) and the minimum distance between the selected elements (Max-Min). This approach results in the Bi-Objective Diversity Problem (BODP). We have shown that a multi-objective optimization in this context provides a set of non-dominated solutions whose diversity could be very useful to a decision maker.

Our main contribution is to propose and to study the performance of six algorithms belonging to three different classes of heuristic methodologies: NSGA-II and SPEA2 from the class of population-based algorithms, GRASP and Iterated Greedy from the class of construction-based algorithms, and Tabu Search and VNS from the class of trajectory-based algorithms. We have adapted all the algorithms to the BODP proposing a particular codification and genetic operators in the case of NSGA-II and SPEA2, and three constructive methods and three local search strategies that are applied in the rest of the algorithms.

In order to determine the value of the parameters of the algorithms, we have run iRace for a selection of the instances under study. Then, we have executed the algorithm with the given configuration for the complete set of the **GKD** instances. The assessment of the results has been made by means of three quality measures that are typically considered in multi-objective evaluation: hypervolume, set coverage and epsilon indicator. These metrics have shown

that the best performer algorithm is the Tabu Search. Regarding the efficiency of the methods, the lower execution times are obtained also by the Tabu Search. This method takes advantage of the local search performance in terms of quality and, at the same time, reduces its execution time due to the tabu list. This controlled intensification is not part of the rest of the proposed methods which, as in the case of Tabu Search, follow the standard implementation.

At this point we are confident that many other algorithmic proposals could be applied to the bi-objective diversity problem, which could be more efficient in the generation of equivalent (or better) results. Therefore, we will extend this study applying different multi-objective optimization techniques over a larger set of instances. In addition, we will conduct new preliminary experiments to explore other combinations of the objective functions. Considering that the maximization of diversity has received much attention in the last years, we can anticipate that this paper, as the first multi-objective approach, will trigger the proposal of new solving methods.

Acknowledgements

This research has been partially supported by the Ministerio de Economía y Competitividad of Spain (Grant Refs. TIN2015-65460-C2 and TIN2014-54806-R).

References

Ağca, S., Eksioglu, B., and Ghosh, J. B. (2000). Lagrangian solution of maximum dispersion problems. *Naval Research Logistics (NRL)*, 47(2):97–114.

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Duarte, A. and Martí, R. (2007). Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 178(1):71 – 84.

Duarte, A., Pantrigo, J. J., Pardo, E. G., and Mladenovic, N. (2015). Multi-objective variable neighborhood search: an application to combinatorial optimization problems. *Journal of Global Optimization*, 63(3):515–536.

Dubois-Lacoste, J. (2014). *Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration*. PhD thesis, Ecole Polytechnique de Bruxelles.

Eiben, A. E. and Smith, J. E. (2003). Introduction to evolutionary computing.

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

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

Feo, T. and Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71.

Ghosh, J. B. (1996). Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175 – 181.

Glover, F. (1992). Formulation and illustrative use of the diversity problem in genetic resource management. *Working Paper, Graduate School of Business Administration, University of Colorado at Boulder*.

Glover, F. (1996). Tabu search and adaptive memory programing advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer.

Glover, F., Ching-Chung, K., and Dhir, K. S. (1995). A discrete optimization model for preserving biological diversity. *Applied Mathematical Modelling*, 19(11):696 – 701.

Glover, F., Kuo, C.-C., and Dhir, K. S. (1998). Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

Hansen, P. and Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449 – 467.

Hassin, R., Rubinstein, S., and Tamir, A. (1997). Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133 – 137.

Jackson, J. (1991). Campuses debate multiculturalism. *Focus*, page 45.

Keely, A. (1989). Maxi-niching the way to a strong brand: Positioning according to systemic dynamics. *Journal of Product Innovation Managment*, 6(3):202206.

Kincaid, R. K. (1992). Good solutions to discrete noxious location problems via metaheuristics. *Annals of Operations Research*, 40(1):265–281.

Knowles, J., Thiele, L., and Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. *TIK-Report No. 214, Computer Engineering and Networks Laboratory, ETH Zurich*.

Kuo, C.-C., Glover, F., and Dhir, K. S. (1993). Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185.

Laguna, M. and Martí, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52.

López-Ibánez, M., Dubois-Lacoste, J., Cáceres, L. P., Stutzle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Martí, R., Campos, V., Resende, M. G., and Duarte, A. (2015). Multi-objective GRASP with path relinking. *European Journal of Operational Research*, 240:54–71.

Martí, R., Gallego, M., and Duarte, A. (2010). A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, 200(1):36 – 44.

Martí, R., Velarde, J. L. G., and Duarte, A. (2009). Heuristics for the bi-objective path dissimilarity problem. *Computers & Operations Research*, 36(11):2905 – 2912.

McConnell, S. (1988). The new battle over immigration. *Fortune*, 117(10):89102.

Page, S. (2007). *The Difference: How the Power of Diversity Creates Better Groups, Firms*. Princeton University Press.

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

Porter, W. M., Rawal, K. M., Rachie, K. O., Wien, H. C., and Williams, R. C. (1975). Cowpea germplasm catalog no 1. *International Institute of Tropical Agriculture*.

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

Ramirez, A. (1979). Cream of the crop? *Wall Street Journal*, 1(29):71 – 84.

Ravi, S. S., Rosenkrantz, D. J., and Tayi, G. K. (1994). Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310.

Resende, M. G. C., Martí, R., Gallego, M., and Duarte, A. (2010). GRASP and path relinking for the max-min diversity problem. *Comput. Oper. Res.*, 37(3):498–508.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033 – 2049.

Sandoya, F., Martínez-Gavara, A., Aceves, R., Duarte, A., and Martí, R. (2018). *Diversity and Equity Models*, pages 1–20. Springer International Publishing, Cham.

Sayyad, A. and Ammar, H. (2013). Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, pages 21–27.

Swierenga, R. (1977). Ethnicity in historical perspective. *Social Science*, 52(1):3144.

Thomas, R. (1990). From affirmative action to affirming diversity. *Harvard Business Review*, 68(2):107117.

Unkel, W. (1985). Natural diversity and national forest planning. *Natural Areas Journal*, 5(10):89102.

von Ghyezy, T. (1986). Product diversity and proliferation as a new mode of competing in the motor car. *Internation Journal of Technology Managment*, 1(3):515518.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195.

Zitzler, E., Laumanns, M., and Thiele., L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece*, pages 95–100.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *Trans. Evol. Comp*, 7(2):117–132.