Improving Search Efficiency and Diversity of Solutions in Multiobjective Binary Optimization by Using Metaheuristics plus Integer Linear Programming

Miguel Ángel Domínguez-Ríos, Francisco Chicano[0000−0003−1259−2990], and Enrique Alba[0000−0002−5520−8875]

ITIS Software, Universidad de Málaga, Spain[⋆]
`miguel.angel.dominguez.rios@uma.es, chicano@lcc.uma.es, eat@lcc.uma.es`

Abstract. Metaheuristics for solving multiobjective problems can provide an approximation of the Pareto front in a short time, but can also have difficulties finding feasible solutions in constrained problems. Integer linear programming solvers, on the other hand, are good at finding feasible solutions, but they can require some time to find and guarantee the efficient solutions of the problem. In this work we combine these two ideas to propose a hybrid algorithm mixing an exploration heuristic for multiobjective optimization with integer linear programming to solve multiobjective problems with binary variables and linear constraints. The algorithm has been designed to provide an approximation of the Pareto front that is well-spread throughout the objective space. In order to check the performance, we compare it with three popular metaheuristics using two benchmarks of multiobjective binary constrained problems. The results show that the proposed approach provides better performance than the baseline algorithms in terms of number of the solutions, hypervolume, generational distance, inverted generational distance, and the additive epsilon indicator.

Keywords: multiobjective optimization · hybrid algorithms · integer linear programming

1 Introduction

Metaheuristics and, in particular, evolutionary algorithms have been very successful solving multiobjective optimization problems using only the information

⋆ This research is partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under contract TIN2017-88213-R (6city); Universidad de Málaga, Consejería de Economía y Conocimiento de la Junta de Andalucía and FEDER under grant number UMA18-FEDERJA-003 (PRECOG); Spanish Ministry of Science, Innovation and Universities and FEDER under contracts RTC-2017-6714-5 (Eco-IoT) and RED2018-102472-T (SEBASENet 2.0); and TAILOR ICT-48 Network (No 952215) funded by EU Horizon 2020 research and innovation programme.

about the fitness function and the constraint violations [18]. Most of the time, in a real context, we have more information than just the evaluation of each solution. The structure of the objective and constraint functions is usually available to be exploited.

With an increasing number of constraints, finding feasible solutions can be difficult for evolutionary computation, where, in many cases, the strategy to get feasibility is based on some kind of penalty: either in the objective function or during the selection or replacement of the solutions in the population [21]. We propose the use of integer linear programming (ILP) solvers for this purpose. The combination of metaheuristics and ILP solvers is not new. The term 'matheuristic' is also used for these hybrids [5] and a great number of papers on the topic have been published [2]. One prominent example is Construct, Merge, Solve and Adapt (CMSA), by Blum et al. [3,4]. In a matheuristic, the ILP solver is commonly used to optimize some subproblem for which an optimal solution can be found in a short time. This differs with our proposal here in which we propose the use of ILP to find a feasible solution that is also located in a region of the objective space determined by a high level strategy to find well-spread solutions (exploration). We leave the optimization to a local search (exploitation). Finding a well-located feasible solution is much easier than optimizing a constrained problem, and the ILP solver is able to do it in a very short time, improving the efficiency of the search. In short, the contributions of this work are:

- The use of ILP solvers to find feasible solutions in particular regions of the objective space very fast for linear multiobjective constrained binary optimization problems.
- The combination of ILP solvers with a high level exploration technique and an efficient local search based on delta-evaluation.
- We compare our algorithm (MultiObjective search based on integer linear programming for Feasibility and Local Search, MOFeLS), with three well-known evolutionary algorithms used in the literature: NSGA-II [9], SPEA2 [25] and MOEA/D [23]. We use 28 problem instances from two different benchmarks.

The method is able to approximate the Pareto front for multiobjective constrained binary optimization problems, including equality constraints, which are a handicap in classical metaheuristics based on bit-flip mutation.

The rest of this paper is organized as follows. In Section 2 we present the basic definitions required to describe our proposed algorithm, which is presented in Section 3. Section 4 presents the computational experiments and Section 5 concludes the paper.

2 Background

In this section, we present the background for this work. It is divided into two subsections: the definition of the general concepts and a general framework to solve MultiObjective Combinatorial Optimization (MOCO) problems.

2.1 Definitions

A binary linear multiobjective program of dimension p is defined as $\min_{x \in X} Cx$, where $C \in \mathbb{R}^{p \times n}$ is the objective matrix in which row i represents the coefficient vector for objective function $f_i(x)$. Vector $x \in \mathbb{B}^n$ is the binary decision vector, and $X = \{x \in \mathbb{B}^n : Ax *_{op} b\}$ is the feasible set, which is supposed to be non-empty. Here, A is an $m \times n$ matrix with the coefficients of the m constraints and $b \in \mathbb{R}^m$ is the right-hand side vector. All elements in C, A and b are real numbers. The operator vector $*_{op}$ has length m, and element i contains the sense of the i-th constraint: '$\leq$', '$\geq$' or '$=$'. The objective matrix is also expressed as $Cx = f(x) = (f_1(x), \ldots, f_p(x))$. All objective functions are considered to be minimized. If we need to maximize some objective, we use the property $max(f_i(x)) = -min(-f_i(x))$.

Given two vectors x and y, we say that y *dominates* x ($y \prec x$), if $f_i(y) \leq f_i(x) \quad \forall i = 1, \ldots, p$, and the inequality is strict for at least one index. When a feasible solution is not dominated by any other feasible solution, we say that it is *efficient*. The image of an efficient solution x, is called a *non-dominated point*, $z = f(x)$. The set of all efficient solutions is called *efficient set*, X_E, and its image is called *Pareto front*, $PF = f(X_E)$. Due to the fact that many of the elements in X_E could lead to the same image, we are only interested in the set PF and one anti-image for each element of this set. Although it is common to use the term *efficient solution* in the decision space and *non-dominated point* in the objective space, sometimes the term *solution* is used to refer to both spaces. Given two p-dimensional vectors l and u with $l < u$, that is, $l_i < u_i, \forall i = 1, \ldots, p$, we define the *box* $[l, u] = \{x \in \mathbb{R}^p \mid l_i \leq x < u_i, \quad \forall i = 1, \ldots, p\}$.

2.2 A framework to solve MOCO problems

In this section we describe a generic and exact formulation for solving MOCO problems. This framework is extracted from the work of Dächert and Klamroth [8]. Our work is based on this approach as we will see in detail in Section 3. The idea of the method is to maintain a set of search zones, $\mathfrak{U}$, which are p-dimensional boxes. Every box is defined by its upper bound and the lower bound is assumed to be the ideal point I of the Pareto front.

In Algorithm 1, $\mathfrak{U}$ is the set of boxes to be analyzed. Initially, the set of non-dominated solutions N is empty (Line 1) and in Line 2 the set of boxes contains the initial element, U, defined by an upper bound for the nadir point [11]. The algorithm then enters a loop until no box is left for its analysis. In each iteration of the loop it selects one box (Line 4), solves an optimization problem based on the box, and if a new non-dominated point is found, it is saved in N. Every time it finds a new non-dominated point, it updates the set $\mathfrak{U}$ accordingly (Line 7), to prevent repeated solutions in the future. Another goal of the updating procedure is to reduce the number of boxes at each iteration. More specifically, at least box B is extracted from $\mathfrak{U}$. The algorithm ends because in MOCO problems the number of non-dominated points is finite.

Algorithm 1 *General exact method for MOCO problems*

```
1: $N = \emptyset$
2: $\mathfrak{U} \leftarrow \{U\}$
3: while ($\mathfrak{U} \neq \emptyset$) do
4:     Select $B \in \mathfrak{U}$
5:     if (Model $P(B)$ is feasible) then
6:         $N = N \cup \{f(x^*)\}$
7:         Update $\mathfrak{U}$
8:     else
9:         $\mathfrak{U} \leftarrow \mathfrak{U} - \{B\}$
10:    end if
11: end while
12: return $N$
```

After a solution is found, some boxes are split into p new boxes each. The splitting process often generates redundant zones. If we have two boxes $B_1 = [I, u^1], B_2 = [I, u^2]$ with $u^1 \leq u^2$, all potential non-dominated points generated by exploring B_1 could also be generated by exploring B_2, which means that B_1 is redundant. Therefore, a filtering process should be implemented after the split. Klamroth et al. [15] proposed two different algorithms for this purpose. In this work we use one of them, called RE (*redundancy elimination*), which consists in eliminating at each iteration the dominated boxes. For more information about how this filtering process works, see [15]. At the end of the execution, Algorithm 1 returns the set N containing the complete Pareto front.

3 Algorithmic proposal

We present our proposal, MOFeLS, in Algorithm 2, which is able to solve any binary linear multiobjective program. If the objective functions or the constraints are not linear, they can be easily linearized adding new variables for the product of two binary variables and some additional constraints. For example, the product of binary variables x_1x_2 can be replaced by y and constraints $2y \leq x_1 + x_2 \leq y + 1$. The algorithm is based on the framework described in Algorithm 1, and uses an ILP solver to find a feasible solution in each iteration. Then, in the heuristic part of the algorithm, a search using a hill climber is conducted to find a local optima in the objective space. A non-dominated set of solutions N is maintained during the search. The mathematical program to solve at each iteration is the one developed by Chalmet et al. [6]. The method combines parameterization of the objective functions (weighted sum) and the ε-constraint method:

$$\begin{aligned} \min \; & \sum_{k=1}^{p} \lambda_k f_k(x) \\ s.t. \quad & f_k(x) \leq u_k, \quad k = 1, \ldots, p \\ & x \in X. \end{aligned} \tag{1}$$

If we use $\lambda_k > 0 \ \forall k = 1, \ldots, p$, and the model of Eq. (1) has a solution, then it is efficient. Vector $u = (u_1, \ldots, u_p)$ is the upper bound of the considered box. The positive weights combination does not have any influence in the feasibility of the model, and we consider $\lambda_k = 1 \ \forall k$. We denote with $P(u)$ this mathematical program.

Algorithm 2 MOFeLS

```
Input: $TILIM$     // Time limit for an ILP solver call
Output: $N$     // Approximated Pareto front
 1: $N = \emptyset$
 2: $\mathfrak{U} = \emptyset$              // List of boxes
 3: $\delta = (\delta_1, \ldots, \delta_1)$   // $0 < \delta_1 < 1$
 4: Estimate bounds for the problem using linear relaxation: $L_b, U_b$
 5: $\mathfrak{U} \leftarrow ([L_b, U_b])$
 6: while ($\mathfrak{U} \neq \emptyset$) and (not stopping condition) do
 7:     $B \leftarrow$ Select box with the highest volume in $\mathfrak{U}$
 8:     if (Solution is found for $P(B.u - \delta)$ in $TILIM$ time) then
 9:         $x \leftarrow$ Get the solution of $P(B.u - \delta)$
10:         $Slack \leftarrow$ Get the slack vector from the ILP solver
11:         $x \leftarrow Hill\_Climbing(x, Slack)$
12:         $N \leftarrow$ Filter $(N \bigcup \{(x, f(x))\})$
13:         $Update$ $(\mathfrak{U}, f(x))$
14:     else
15:         $\mathfrak{U} \leftarrow \mathfrak{U} - \{B\}$
16:     end if
17: end while
```

The input parameter *TILIM* represents the maximum total time employed by the ILP solver at each call. It is fixed during the execution. At the beginning of Algorithm 2, we initialize the non-dominated set, N, and the list of boxes, $\mathfrak{U}$. The vector δ in Line 3 is necessary to guarantee that the potential new solution is not equal to the upper bound of the box, $B.u$ (see Line 8).

Before starting the loop, we need to determine the initial box. This must contains a lower and an upper bound for the Pareto front. This estimation is done by solving linear relaxations (the decision variables are continuous instead of binary) of $\min\{f_i(x)\}$ and $\max\{f_i(x)\}$. The minimum value obtained for each f_i is the i-th component of the lower bound for the ideal point. The maximum values form the upper bound of the nadir point. We have to note that for hard problems, the execution time used for this calculation could be also high, so if the parameter *TILIM* is very low, the algorithm could end without any solution. After inserting the initial box in the list $\mathfrak{U}$ (Line 5), the algorithm runs a loop while there exists a box to analyze and it does not exceed a preset limited time (stopping condition). At the beginning of the loop, the algorithm selects always the box with the highest volume (Line 7). This is efficiently implemented using heaps for the list $\mathfrak{U}$. Taking the box with the highest volume helps to increase the diversity of solutions in the objective space (spread), since it tends to select boxes where many potential solutions exist far away from other found solutions. This is just a heuristic, and it could happen that no solution is found in the box at all. But experiments show that it works well in general. The upper bound of the analyzed box is the vector u in the model, and MOFeLS then calls the ILP solver to obtain a feasible solution, always under the condition of not exceeding the *TILIM* time. The reason for introducing the *TILIM* parameter is the following: when the ILP solver takes 'long time' to calculate a feasible solution, it is probable that the box is empty (the corresponding objective space region has no solution), so we discard it and do not waste time looking for solutions that may not exist.

The possible outputs after an execution of the ILP solver[1] are:

a) An optimal solution is found.
b) A feasible suboptimal solution is found.
c) *TILIM* exceeded, but a feasible solution is found.
d) There is no solution (infeasible).
e) *TILIM* exceeded and no solution is found.
f) Problem is infeasible or unbounded.

In the cases where the output is d), e) or f), we discard that box (Line 15) and get the next one. Otherwise, we have a new feasible solution and MOFeLS calls the hill climber to improve it. Every local optimum is inserted into $\mathfrak{U}$ and repeated or dominated ones are discarded during the execution (Line 12). The updating procedure splits all boxes affected by the new solution found, reducing the total search space, and filtering the redundant boxes, avoiding repeated solutions in the future. For more information of how the redundancy elimination (RE) works, see [15].

Algorithm 3 *Hill_Climbing*(x, *Slack*)

```
1: Set all components of $x$ unmarked
2: while ($x$ has an unmarked component) do
3:     $j \leftarrow$ Select a random index of an unmarked component of $x$
4:     Mark $x_j$
5:     $(\delta_c, \Delta) \leftarrow Bit\_Flip(x, j, Slack)$
6:     if $(\delta_c < 0)$ then
7:         $x_j = 1 - x_j$
8:         $Slack \leftarrow Slack + \Delta$   // Update vector $Slack$
9:         Unmark all components in $x$
10:    end if
11: end while
12: return $(x)$
```

Next, we explain an efficient strategy to reduce the computational cost in the hill climber, which starts with a feasible solution and applies local search until

[1] Outputs for the ILP solver CPLEX 12.6.2.

a local optimum is found. In this case, the fitness value of a solution is given by $\sum_{i=1}^{p} f_i(x)$ (see Eq. (1)). The neighborhood of a solution x is the set of solutions at Hamming distance 1 from x (one differing variable). The *Hill_Climbing* function is described in Algorithm 3. The input parameters of this function are the feasible solution x, and the slack vector $Slack = b - Ax$. This vector has one real component per constraint in the model. Every value represents the gap between the right-hand side value and the value of the constraint. These values can be extracted from the ILP solver. When the local optimum is found, the modified x is also the output. The hill climber selects every decision variable and checks if making a bit-flip (using the *Bit_Flip* function) maintains feasibility and improves the solution with a decrease in the objective cost ($\delta_c < 0$). If so, it moves to the new solution and starts again exploring the neighborhood, until no 1-bit improvement is found.

We note that in constraints of type '$\leq$' the slack value is always non-negative. In '$\geq$' constraints it is always non-positive, and for equality constraints it must be 0 for any feasible solution. We illustrate this with an example:

$$\min f(x) = (x_2 + x_3, x_3 + x_4 + 2x_5),$$

s.t.

$$\begin{aligned} & x_1 + 2x_2 - 2x_3 + 7x_4 + x_5 \leq 8, \\ & x_1 + x_2 + x_3 - x_4 + 4x_5 \geq 2, \\ & x_1 + x_5 = 2, \\ & x \in \mathbb{B}^5. \end{aligned} \tag{2}$$

Suppose that the ILP solver reports the feasible solution $x = (1, 1, 0, 0, 1)$. Then, $Slack = (4, -4, 0)$ and the objective value is $f_1(x) + f_2(x) = 1 + 2 = 3$.

The *Bit_Flip* function uses delta-evaluation to efficiently compute the objective function and the constraints violation. The input parameters are the decision vector, the index of the variable to flip, and the slack vector. Its pseudocode is displayed in Algorithm 4. At the beginning, we define a vector Δ that controls the variation of *Slack*, and initialize it to the vector $(0, \ldots, 0)$. In Line 3 we get the coefficient in the objective function associated to variable x_j, that is, $\delta_c = \sum_{i=1}^{p} c_{ij}$. If this value equals 0, then no improvement in the objective function is made, regarding the value of x_j. Moreover, in the case of $\delta_c < 0$, if $x_j = 1$, and a bit-flip is done, the variable will not be in the solution, so the objective cost increases $|\delta_c|$ units, and the bit-flip should be discarded. For the same reason, if $\delta_c > 0$ and $x_j = 0$, the bit-flip is discarded. These conditions are condensed in Line 4. If $\delta_c \cdot (-1)^{x_j} < 0$, we continue checking the constraints, and change the value of δ_c to $-|\delta_c|$. That would be the improvement in objective function if the bit-flip is finally accepted. Similar arguments have been carried out for the analysis of the constraints. If one of them is violated, we return $\delta_c = 0$.

Let us illustrate this again with the example of Eq. (2). Starting from the initial solution, $x = (1, 1, 0, 0, 1)$, the objective function equals 3. We analyze

Algorithm 4 *Bit_Flip*(x, j, *Slack*)

```
1: // Analyzing whether flipping $x_j$ affects the objective cost
2: $\Delta = (0, \dots, 0)$        // Variation of $Slack$ vector
3: $\delta_c \leftarrow$ Get objective coefficient cost of $x_j$ from the ILP solver
4: if $(\delta_c \cdot (-1)^{x_j}) \geq 0)$ then return $(0, (0, \dots, 0))$
5: $\delta_c = - \mid \delta_c \mid$
6: // Analyzing constraints
7: for $i = 1$ to $m$ do
8:    $d \leftarrow$ Type of the slack (1 for '$\leq$' ; -1 for '$\geq$' ; 0 for '=')
9:    if $(d \neq 0)$ then
10:       $t \leftarrow a_{ij} \cdot (-1)^{x_j}$
11:       if $(d \cdot Slack[i] < d \cdot t)$ then return $(0, (0, \dots, 0))$
12:       $\Delta_i = -t$
13:    else
14:       if $(a_{ij} \neq 0)$ return $0, (0, \dots, 0)))$
15:    end if
16: end for
17: return $(\delta_c, \Delta)$
```

whether a bit-flip on the second component is possible. We have $x_2 = 1$ and $\delta_c = 1$. The condition in Line 4 is false and δ_c is changed to $\delta_c = -1$ (Line 5). We continue analyzing the constraints. For the first one, $d = 1$, $t = 2 \cdot (-1) = -2$ and it holds $d \cdot Slack[1] = 1 \cdot 4 \geq -2$. The constraint is not violated (Line 11), and Δ_1 changes to $\Delta_1 = 2$. This means that the gap for that constraint will increase in two units if we finally accept the flip. In the second constraint, $Slack[2] = -4$, $d = -1$, $t = 1 \cdot (-1) = -1$ and it holds $(-1) \cdot (-4) \geq (-1) \cdot (-1)$. This constraint is not violated and Δ_2 changes to $\Delta_2 = 1$. This means that flipping the variable x_2 makes the slack on the second constraint to increase in one unit (we are nearer to the saturation of the constraint). For the last constraint, $d = 0$ (equality constraint) and $a_{ij} = 0$, so finally the function returns $(-1, (2, 1, 0))$ and the flip for x_2 is accepted. When we return to the *Hill_Climbing* function, we update the slack vector, and $Slack = (4 + 2, -4 + 1, 0 + 0) = (6, -3, 0)$. The new objective value is $3 - 1 = 2$.

4 Computational experiments

We conduct the experimental study in this section. First, we present the instances used in this work. In the second subsection, we define the metrics we use to assess the results. In the third, we set the input parameters of all the used algorithms and, finally, we provide the numerical results.

4.1 Instances

A representative benchmark of 28 binary linear multiobjective instances are selected from two existing benchmarks in the literature. We select 20 instances[2] from the work of Kirlik et al. [14], and 8 instances from multiobjective multidimensional knapsack problems[3]. We summarize them in the following:

- 10 instances of a 3-dimensional assignment problem, each with 50 agents and tasks. Every instance has 2500 variables and 100 constraints. The group of all of them is named AP.
- 10 instances of a 1-dimensional knapsack with 4 objectives. Every instance has 40 variables and one constraint. The group of all of them is named KP.
- 4 instances of a 3-dimensional knapsack problem with 3 objectives, with 100, 250, 500 and 750 variables, respectively. They are named MKP3_100, MKP3_250, MKP3_500 and MKP3_750.
- 4 instances of a 4-dimensional knapsack problem with 4 objectives, with 100, 250, 500 and 750 variables, named MKP4_100 to MKP4_750.

Multiobjective assignment problems have been selected because they have equality constraints. In addition, knapsack problems with different numbers of knapsacks and objectives are chosen because they are widely used NP-hard problems.

4.2 Quality indicators

To evaluate the quality of the solutions [27] in our algorithm, we have decided to use a group of quality indicators that are representative [13]. The first of them is the *overall non-dominated vector generation*, which is defined as the cardinal of the elements found in the objective set, after discarding the dominated vectors,

$$\mathrm{ONVG}(N) = |N|. \tag{3}$$

The *hypervolume* indicator is the quality measure with the highest discriminatory power among the known unary quality measures [19, 26, 27]. There are many software packages that calculate the hypervolume of a set, given a reference point, as in the works of Fonseca et al. [12] and While et al. [22]. Given a set of k points in the objective space, $N = \{z^1, z^2, \dots, z^k\}$, the *hypervolume* HV is the measure of the region which is simultaneously dominated by N and bounded by a reference point $r \in \mathbb{R}^p$. It can be expressed by

$$\mathrm{HV}(N, r) = \mathit{volume}\left(\bigcup_{j=1}^{k} [z^j, r]\right). \tag{4}$$

The reference point can be taken as $r_i = \max_{j=1,\dots,k} z_i^j \ \forall i = 1, \dots, p$. This is a good choice when we have no information about the complete Pareto front.

[2] Available in http://home.ku.edu.tr/∼moolibrary/

[3] Available in https://sop.tik.ee.ethz.ch/download/supplementary/testProblemSuite/

In some cases, a positive value for each component of the reference point is added [16], in order to take into account the extreme solutions.

The *generational distance* [9] is frequently used in multiobjective evolutionary algorithms and it is defined as

$$\mathrm{GD}(N,P) = \frac{\left(\sum_{i=1}^{|N|} d_i^2\right)^{1/2}}{|N|}, \tag{5}$$

where P is the reference set and d_i is the smallest Euclidean distance from a vector in N to the closest vector in P. The reference set we use depends on the instance. In those instances in which we have the complete Pareto front, we use the Pareto front as reference set and the metrics are more precise. For the remaining instances, the reference set consists of the union of all the outputs of the independent runs of the algorithms plus the union of the outputs of each algorithm after 40 minutes of computation. These unions are filtered in order to have only non-dominated points.

The *inverted generational distance* has a similar formulation to GD, but in this case we take the smallest distance for every element in P to the closest solution in N,

$$\mathrm{IGD}(N,P) = \mathrm{GD}(P,N). \tag{6}$$

The *additive epsilon indicator* gives the minimum additive factor by which the approximation set has to be translated in the objective space in order to weakly dominate the reference set [17, 27]. We have scaled each objective to obtain a value in the range [0,1]. This additive epsilon indicator is defined as

$$\varepsilon_+(N,P) = \max_{x\in P} \min_{y\in N} \max_{i=1,\dots,p} \left(\frac{y_i - x_i}{r_i}\right), \tag{7}$$

where r_i is the range of objective i in N.

Note that all the metrics used in this paper are Pareto compliant [24].

4.3 Parameters of the algorithms

Before the execution of the algorithms we use the *iterated racing for automatic algorithm configuration* (IRACE) [20] to tune and obtain the best parameter values for each algorithm. For MOFeLS algorithm, the *TILIM* parameter varies in the set $\{0.0001, 0.001, 0.01, 0.1, 0.2, 0.4, 0.6, 0.8, 1\}$. IRACE chose the value 0.4. For the other three metaheuristics we let IRACE to decide the best configuration between the following ranges: the number of cut points in the crossover varies from 1 to 5; the crossover probability is free in the set $[0, 1]$; the mutation probability is in $[0, 0.5]$; and the population size varies from 10 to 1000 with a step of 10. For the constrained MOEA/D algorithm [1], we also set the probability of selecting the solution in the neighborhood (solutions with close weights) or in the whole population, in the range $[0, 1]$, and the neighborhood size varies from

Table 1. Configuration of the parameters of NSGA-II, SPEA2 and MOEA/D using IRACE.

Algorithm	Number of crossover cuts	Crossover probability	Mutation probability	Population size	Neighborhood probability	Neighborhood size
NSGA-II	5	0.77	0.0009	310	-	-
SPEA2	4	0.70	0.0017	430	-	-
MOEA/D	2	0.86	0.0036	760	0.86	15

2 to 20. The final configuration computed by IRACE in the three metaheuristics is shown in Table 1.

The ILP solver used in MOFeLS is CPLEX 12.6.2. We changed three CPLEX parameters. The first one, CPX_PARAM_MIPEMPHASIS, controls trade-offs between speed, feasibility, optimality, and moving bounds. Setting the value to 1, we emphasize feasibility over optimality. This is done because we only use the solver to obtain a feasible solution. CPX_PARAM_INTSOLLIM sets the number of integer solutions to be found before stopping. We set this parameter to 1 (we only need one feasible solution). Finally, CPX_PARAM_TILIM sets the maximum time, in seconds, for a call to the optimizer (denoted as *TILIM* in the the paper). This parameter was tuned with IRACE, as mentioned above, and it is set to 0.4.

4.4 Numerical results

For the 28 instances considered, we execute 30 times each algorithm, using a cluster with ten machines Intel Core 2 Quad (Q9400) CPU at 2.7 GHz, a total of 4 cores each, 11 GB of memory and Ubuntu 16.04 LTS. For each run we used only 1 core, 2 GB of RAM and 30 seconds of computation, and reported the average values for the quality indicators. In the groups AP and KP we have also considered average values among the ten instances of each group. MOFeLS[4] is programmed using C++. For NSGA-II, SPEA2 and the constrained MOEA/D, we used the jMetal 5.8 implementation[5] (see jMetal package in [10]), conveniently modified to use a time limit as stopping condition.

When executing the three metaheuristics, only solutions which do not violate any constraint were considered, and an ulterior filtering of the points were made, avoiding repetition or domination between them. This is done off-line. We ran an exact algorithm during one week and we divided the instances into two groups: those for which we obtained the complete Pareto front using the exact algorithm and the remaining ones. In Section 4.2 we explained how the reference set was computed in each case. In Tables 2 and 3 we show the numerical results for these two groups of instances, respectively.

We can observe in the numerical results of Table 2 that none of the three classic metaheuristics is able to find any solution for AP. This is because those algorithms use random bit-flip mutation, and if one bit is changed, the constraints

[4] https://github.com/MiguelAngelDominguezRios/MOFeLS
[5] https://github.com/jMetal/jMetal

Table 2. Computational results for instances with known Pareto front. Best results are marked in bold. Undefined values are marked with hyphen.

		MOFeLS	NSGA-II	SPEA2	MOEA/D
ONVG	AP	**98.85**	0.00	0.00	0.00
	KP	**628.62**	97.40	119.46	122.41
	MKP3_100	**225.70**	40.03	42.27	94.03
HV	AP	**9.90E+07**	0.00E+00	0.00E+00	0.00E+00
	KP	**4.56E+14**	3.70E+14	3.72E+14	4.38E+14
	MKP3_100	**9.41E+11**	6.04E+11	6.52E+11	**9.41E+11**
IGD	AP	**0.54**	-	-	-
	KP	**6.57**	27.83	27.169	15.88
	MKP3_100	13.36	44.60	38.57	**12.18**
GD	AP	**1.94**	-	-	-
	KP	**0.78**	16.17	11.55	5.91
	MKP3_100	**8.55**	51.27	62.65	28.76
ε_+	AP	**0.11**	-	-	-
	KP	**0.11**	0.49	0.49	0.27
	MKP3_100	**0.22**	0.58	0.56	0.31

can be violated with high probability. One of the main advantage of MOFeLS is that it works well with equality constraints because the ILP solver provides a feasible solution. When the local search is not able to find an improving move due to the equality constraints, the feasible solution provided by the ILP solver is used as the final solution. For the rest of instances in the table, we observe that MOFeLS finds the best values for all the instances except in one case for IGD. Note that a lower average value in GD is preferable, meaning that the approximated front is closer to the Pareto front.

For the other group of instances, we do not have the complete Pareto fronts, and we execute during 40 minutes each algorithm and take as a reference set the union of these sets plus the union of the corresponding executions at 30 s. This is done for each instance. Analyzing Table 3, we see that MOFeLS has the best average number of solutions with a great difference with respect to the others. In this case, the value of the metric is just an approximation because we do not know the Pareto front. We see that MOFeLS has also the best value for HV, IGD and GD in all the cases. For the ε_+ metric, there is no clear winner and we need to do the corresponding hypothesis tests to support any conclusion. The number of solutions in MOEA/D is lower than in MOFeLS. This means that fewer solutions are closer to the reference Pareto set and this facilitates a lower value for ε_+ indicator.

To justify the well-spread of the solutions given by MOFeLS, we first define the percentage of the total hypervolume reached,

$$\mathrm{HVR}(N, r) = \frac{\mathrm{HV}(N, r)}{\mathrm{HV}(PF, r)}. \tag{8}$$

Table 3. Computational results for instances with unknown Pareto front. Best results are marked in bold.

		MOFeLS	NSGA-II	SPEA2	MOEA/D
ONVG	MKP3_250	**438.63**	43.60	48.20	160.47
	MKP3_500	**430.33**	59.97	49.00	194.87
	MKP3_750	**283.87**	47.87	40.27	165.67
	MKP4_100	**393.03**	91.10	107.83	168.77
	MKP4_250	**546.03**	93.63	100.33	267.80
	MKP4_500	**384.50**	91.53	95.50	292.93
	MKP4_750	**307.67**	84.17	82.57	240.77
HV	MKP3_250	**5.25E+12**	2.63E+11	4.42E+11	3.79E+12
	MKP3_500	**6.31E+13**	2.02E+12	1.93E+12	1.96E+13
	MKP3_750	**1.89E+14**	4.13E+12	3.98E+12	2.48E+13
	MKP4_100	**8.06E+15**	6.5E+13	2.36E+14	2.55E+15
	MKP4_250	**2.32E+17**	1.77E+15	3.35E+15	4.12E+16
	MKP4_500	**5.57E+18**	2.08E+16	3.06E+16	3.18E+17
	MKP4_750	**1.69E+19**	4.88E+16	6.62E+16	3.56E+17
IGD	MKP3_250	**32.65**	86.43	84.88	43.47
	MKP3_500	**51.91**	132.16	176.40	110.91
	MKP3_750	**75.17**	255.64	354.76	268.28
	MKP4_100	**17.26**	39.69	32.97	19.20
	MKP4_250	**31.00**	69.93	73.86	48.40
	MKP4_500	**53.25**	129.94	175.09	126.95
	MKP4_750	**71.05**	254.02	356.08	266.17
GD	MKP3_250	**8.56**	519.09	615.60	138.02
	MKP3_500	**40.18**	1487.41	2369.15	732.73
	MKP3_750	**106.41**	3592.33	5588.16	1985.89
	MKP4_100	**31.22**	49.77	72.07	47.02
	MKP4_250	**60.16**	337.82	478.08	183.39
	MKP4_500	**280.19**	1409.77	2112.19	897.70
	MKP4_750	**396.11**	3167.34	4561.25	2008.69
ε_+	MKP3_250	0.26	0.41	0.35	**0.20**
	MKP3_500	0.19	0.33	0.27	**0.18**
	MKP3_750	0.23	0.36	0.27	**0.22**
	MKP4_100	**0.15**	0.47	0.42	0.24
	MKP4_250	**0.12**	0.44	0.40	0.30
	MKP4_500	**0.14**	0.42	0.35	0.32
	MKP4_750	**0.10**	0.33	0.24	0.24

We show in Table 4 the average values of HVR and IGD for every instance for which we know the Pareto front. Thus, a value close to 1 in HVR indicates a good spread over the objective space. On the other hand, IGD measures the average distances between points in the Pareto front to the closest points in N. The more close is this value to 0, the more well-spread is the approximated Pareto front.

To finish our computational experiments, we do the appropriate hypothesis tests to support the conclusions. For each metric, algorithm and instance we

Table 4. Average values for HVR and IGD using MOFeLS for each instance with known Pareto front.

	HVR	IGD		HVR	IGD
AP1	0.8511	0.53	*KP1*	0.9953	9.37
AP2	0.8549	0.56	*KP2*	0.9785	6.06
AP3	0.8562	0.57	*KP3*	0.9999	4.56
AP4	0.8494	0.55	*KP4*	0.9957	8.28
AP5	0.8666	0.53	*KP5*	0.9867	7.21
AP6	0.8620	0.56	*KP6*	0.9908	6.98
AP7	0.8635	0.56	*KP7*	0.9947	6.64
AP8	0.8533	0.47	*KP8*	0.9994	5.30
AP9	0.8636	0.51	*KP9*	0.9819	6.77
AP10	0.8770	0.52	*KP10*	0.9999	4.52
MKP3_100	0.9041	13.36			

compute the average values given by the 30 runs of the algorithm. Then, for each metric and algorithm we use the 28 samples (average of runs for each instance) as an input for the non-parametric Wilcoxon signed rank test. We compare MOFeLS with the other three using significance level $\alpha = 0.01$. All the p-values are below 10^{-5}, leading us to the conclusion that MOFeLS is the best overall algorithm for all the metrics. We also conclude, based on the results of the HVR and IGD indicators, that MOFeLS finds a well-spread set of solutions over the objective space.

5 Conclusions and future work

We present a hybrid algorithm able to solve binary linear constrained multiobjective problems, with good spread of the solutions over the objective space. The algorithm is able to deal with equality constraints, which is a handicap using other metaheuristics. MOFeLS provides the best approximated Pareto fronts in terms of number of solutions, hypervolume, generational distance and inverted generational distance in all the cases. For the metric ε_+, the best results are obtained by our proposed algorithm and MOEA/D. We think these preliminary results encourage the investigation of other hybrids combining exact methods and metaheuristics. In the future, it would be interesting to compare this algorithm with other matheuristics existing in the literature using additional benchmark problems. We also plan to extend the research and consider higher order objective and constraint functions (not only linear), using the advances in gray-box optimization for constrained multiobjective problems [7]. We also want to analyze the behaviour of the algorithms as time progresses.

References

1. Asafuddoula, M., Ray, T., Sarker, R., Alam, K.: An adaptive constraint handling approach embedded moea/d. In: 2012 IEEE Congress on Evolutionary Computa-

tion. pp. 1–8. IEEE (2012)
2. Ball, M.O.: Heuristics based on mathematical programming. Surveys in Operations Research and Management Science **16**(1), 21–38 (2011)
3. Blum, C., Pereira, J.: Extension of the cmsa algorithm: an lp-based way for reducing sub-instances. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 285–292 (2016)
4. Blum, C., Pinacho, P., López-Ibáñez, M., Lozano, J.A.: Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. Computers & Operations Research **68**, 75–88 (2016)
5. Boschetti, M.A., Maniezzo, V., Roffilli, M., Röhler, A.B.: Matheuristics: Optimization, simulation and control. In: International Workshop on Hybrid Metaheuristics. pp. 171–177. Springer (2009)
6. Chalmet, L., Lemonidis, L., Elzinga, D.: An algorithm for the bi-criterion integer programming problem. European Journal of Operational Research **25**(2), 292–300 (1986), https://doi.org/10.1016/0377-2217(86)90093-7
7. Chicano, F., Whitley, D., Tinos, R.: Efficient hill climber for constrained pseudo-boolean optimization problems. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 309–316 (2016)
8. Dächert, K., Klamroth, K.: A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. Journal of Global Optimization **61**(4), 643–676 (2015), https://doi.org/10.1007/s10898-014-0205-z
9. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: International conference on parallel problem solving from nature. pp. 849–858. Springer (2000)
10. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. Advances in Engineering Software **42**(10), 760–771 (2011)
11. Ehrgott, M., Tenfelde-Podehl, D.: Computation of ideal and nadir values and implications for their use in MCDM methods. European Journal of Operational Research **151**(1), 119–139 (2003), https://doi.org/10.1016/S0377-2217(02)00595-7
12. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: Proceedings of the IEEE Congress on Evolutionary Computation, 2006. CEC 2006. pp. 1157–1163. IEEE (2006), https://doi.org/10.1109/CEC.2006.1688440
13. Jiang, S., Ong, Y.S., Zhang, J., Feng, L.: Consistencies and contradictions of performance metrics in multiobjective optimization. IEEE transactions on cybernetics **44**(12), 2391–2404 (2014), https://doi.org/10.1109/TCYB.2014.2307319
14. Kirlik, G., Sayın, S.: A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. European Journal of Operational Research **232**(3), 479–488 (2014), https://doi.org/10.1016/j.ejor.2013.08.001
15. Klamroth, K., Lacour, R., Vanderpooten, D.: On the representation of the search region in multi-objective optimization. European Journal of Operational Research **245**(3), 767–778 (2015), https://doi.org/10.1016/j.ejor.2015.03.031
16. Li, M., Yao, X.: Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Computing Surveys **52**(2), 1–38 (2019)
17. Liefooghe, A., Derbel, B.: A correlation analysis of set quality indicator values in multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 581–588 (2016)
18. Liu, Q., Li, X., Liu, H., Guo, Z.: Multi-objective metaheuristics for discrete optimization problems: A review of the state-of-the-art. Applied Soft Computing **93**, 106382 (2020). https://doi.org/https://doi.org/10.1016/j.asoc.2020.106382

19. López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. European Journal of Operational Research **235**(3), 569–582 (2014), https://doi.org/10.1016/j.ejor.2013.10.043
20. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
21. Mezura-Montes, E., Coello Coello, C.A.: Constraint-handling in nature-inspired numerical optimization: Past, present and future. Swarm and Evolutionary Computation **1**(4), 173 – 194 (2011). https://doi.org/https://doi.org/10.1016/j.swevo.2011.10.001
22. While, L., Bradstreet, L., Barone, L.: A fast way of calculating exact hypervolumes. IEEE Transactions on Evolutionary Computation **16**(1), 86–95 (2012), https://doi.org/10.1109/TEVC.2010.2077298
23. Zhang, Q., Li, H.: Moea/d: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on evolutionary computation **11**(6), 712–731 (2007)
24. Zitzler, E., Brockhoff, D., Thiele, L.: The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In: International Conference on Evolutionary Multi-Criterion Optimization. pp. 862–876. Springer (2007)
25. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm. TIK-report **103** (2001)
26. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms—a comparative case study. In: International conference on parallel problem solving from nature. pp. 292–301. Springer (1998), https://doi.org/10.1007/BFb0056872
27. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Da Fonseca, V.G.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on evolutionary computation **7**(2), 117–132 (2003), https://doi.org/10.1109/TEVC.2003.810758