A primal heuristic to compute an upper bound set for multi-objective 0-1 linear optimisation problems

Xavier Gandibleux
Université de Nantes
Xavier.Gandibleux@univ-nantes.fr

Guillaume Gasnier
Université de Nantes
Guillaume.Gasnier@etu.univ-nantes.fr

Saïd Hanafi
Université Polytechnique - INSA Hauts-de-France
Said.Hanafi@uphf.fr

ABSTRACT

This paper presents an algorithm aiming to compute an upper bound set for a multi-objective linear optimisation problem with binary variables (p-01LP). Inspired by the well known « Feasibility Pump » algorithm in single objective optimisation, it belongs to the class of primal heuristics. The proposed algorithm, named « Gravity Machine », aims to deal with generic p-01LP. In that sense, it does not exploit any information coming from the combinatorial structure of the problem. Sober in memory and computational resources, and given a time limit, the algorithm explores a p-01LP problem in order to discover a tight upper bound set, without guarantee of returning such a set. The paper describes the current version of the proposed algorithm. It reports numerical results obtained on instances of set partitioning problem available on the OR-library, extended to two objectives.

KEYWORDS

Multi-Objective Optimisation, 01 Linear Programming, Bound Sets, Primal Heuristic.

1 INTRODUCTION

1.1 Definitions and Notations

The multi-objective 0-1 linear optimisation problems with p objectives (p-01LP) considered in this paper can be formulated as follows:

$$\begin{array}{rcl} \min z(x) & = & Cx \\ \text{subject to } Ax & \leqq & b \\ x & \in & \{0,1\}^n \end{array}$$

where

- $x \in \{0,1\}^n$, the vector of n binary variables x_j, $j = 1, \ldots, n$;
- $A \in \mathbb{R}^{m \times n}$, the m constraints $A_i x \leq b_i$, $i = 1, \ldots, m$ and $b \in \mathbb{R}^m$;
- $C \in \mathbb{R}^{p \times n}$, the objective matrix where $p \geq 2$;
- $X := \{x \in \{0,1\}^n \mid Ax \leqq b\} \subseteq \mathbb{R}^n$, the set of feasible solutions, with $\mathbb{R}^n$ the decision space;
- $Y := \{Cx \mid x \in X\} \subseteq \mathbb{R}^p$, the outcome set, with $\mathbb{R}^p$ the objective space.

1.2 Solutions and performance vectors

We assume that no feasible solution minimizes all p objectives simultaneously and use the following notation for componentwise orders in $\mathbb{R}^p$. Let $y, y' \in \mathbb{R}^p$, we denote $y \leqq y'$ (y *weakly dominates* y') if $y_k \leqq y'_k$ for $k = 1, \ldots, p$, $y \leq y'$ (y *dominates* y') if $y \leqq y'$ and $y \neq y'$, and $y < y'$ (y *strictly dominates* y') if $y_k < y'_k, k = 1, \ldots, p$. We define $\mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p : y \geqq 0\}$ and analogously $\mathbb{R}^p_{\geq}$ and $\mathbb{R}^p_{>}$.

Let $z(x) = (z^1(x), \ldots, z^p(x)) \in \mathbb{R}^p$ be the objective vector, also named point or performance, associated with a solution $x \in \mathbb{R}^n$. A feasible solution $\hat{x} \in X$ is called *efficient (weakly efficient)* if there does not exist $x \in X$ such that $z(x) \leq z(\hat{x})$ ($z(x) < z(\hat{x})$). If $\hat{x}$ is (weakly) efficient, then $z(\hat{x})$ is called *(weakly) nondominated*. The *efficient set* $X_E \subseteq X$ is defined as $X_E := \{x \in X : \nexists\, x' \in X : z(x') \leq z(x)\}$, and its image under the vector-valued linear mapping C is referred to as the *nondominated set* $Y_N := \{z(x) \mid x \in X_E\}$. Equivalently, Y_N can be defined by $Y_N := \{y \in Y : (y - \mathbb{R}^p_{\geqq}) \cap Y = \{y\}\}$, (See [3] for further details.).

1.3 Bound sets

Branch and bound is a well-known generic method for computing an optimal solution of a single objective optimization problem. Based on the "divide to conquer" idea, it consists in an implicit enumeration principle, viewed as a tree search. Branch and bound has been extended to deal with multi-objective optimisation problems, see [23] for a recent state-of-the-art. Nevertheless, the contributions on the extensions of the components of branch and bound for multi-objective optimization are recent. For example, the concept of bound sets, which extends the classic notion of bounds, has been mentioned by Villarreal and Karwan in 1981. But it was only developed for the first time in 2001 by Ehrgott and Gandibleux [5], and fully defined in 2007 [7].

For a minimisation problem, we denote by

- U, an *upper bound set* (also named *primal bound set*) on Y_N,
- L, a *lower bound set* (also named *dual bound set*) on Y_N.

For a sake of simplicity, only the definition presented in 2001 for U on Y_N is presented here (the reader will find the required terminology, the refined definitions and properties about bound sets in [7]). In this definition, lower and upper bound sets are proposed to bound a subset $Y' \subset Y$ of feasible points.

Definition 1 (A lower bound set according to [5]). *A lower bound set for Y' is a subset $L \subseteq \mathbb{R}^p$ such that*

(i) *For each $y \in Y'$ there is some $l \in L$ such that $l \leqq y$,*
(ii) *There is no pair $y \in Y'$, $l \in L$ such that y dominates l.*

Definition 2 (An upper bound sets according to [5]). *An upper bound set for Y' is a subset $U \subset \mathbb{R}^p$ such that*

(iii) *For each $y \in Y'$ there is some $u \in U$ such that $y \leqq u$,*
(iv) *There is no pair $y \in Y'$, $u \in U$ such that u dominates y.*

The rest of this paper is devoted to a generic primal heuristic aiming to generate a set of feasible solutions $X_U \subseteq X$ such that the

Proc. of the 1st Multi-Objective Decision Making Workshop (MODeM 2021), Hayes, Mannion, Vamplew (eds.), July 14-16, 2021, Online, http://modem2021.cs.nuigalway.ie. 2021.

image $z(X_U) = \{z(x) : x \in X_U\}$ is an upper bound set. In other words, we look for an upper bound set U such that $z^{-1}(U) = \{x \in X : z(x) \in U\} = X_U$ or equivalently $z(X_U) = U$.

1.4 State of the art

A vast number of multi-objective metaheuristics (MOMH) have been proposed (see i.e [6, 8, 16]) since the mid-90s to approximate Y_N by constructing a subset of an approximate set X^* of the efficient set X_E such the image $z(X^*)$ is an upper bound set. They represent a first group of algorithms and are mainly experimented on multi-objective non-linear problems or on multi-objective combinatorial problems, where for these latter the specific structure of the problem is strongly exploited by the algorithms. Nevertheless, relatively few MOMH [18] have been tested on general multi-objective 0-1 linear optimisation problems where by assumption none structure is available to be exploited by the algorithm.

A second group of algorithms is constitued by general primal heuristics, mainly represented by « Feasibility Pump » (FP), a heuristic introduced in 2005 [9] for computing feasible solutions of single-objective integer linear programs. FP may be outlined as follows (see Figure 1). Starting from an optimal solution of the LP-relaxation $\overline{x}^0 \in \overline{X}$, where $\overline{X} = \{x \in [0,1]^n : Ax \leqq b\}$, the FP heuristic generates iteratively two sequences of solutions $\overline{x}^t \in \overline{X}$ and $\tilde{x}^t \in \{0,1\}^n$. A binary solution $\tilde{x}^t$, $t \geq 0$, is obtained from the fractional $\overline{x}^t$ by a rounding procedure (e.g. scalar rounding to the nearest integer), while a fractional solution $\overline{x}^t$, $t > 0$, is an optimal solution of the projection problem (see projectSolution in Section 2.2). Thus, a new fractional solution $\overline{x}^{t+1}$ is generated as the closest feasible LP solution with respect to the solution $\tilde{x}^t$. After some iterations the FP heuristic may start to cycle, i.e., a particular sequence of points $\tilde{x}^t$ and $\tilde{x}^{t'}$, $t' > t$, is visited repetitively. That issue is resolved applying a perturbation to the current solution $\tilde{x}^{t'}$ as soon as a cycle is detected (see perturbSolution in Section 2.2). The processus is repeated until a feasible solution is discovered or a maximum number of iterations or time limit is reached. Today FP is integrated in several MIP solvers such as GLPK, Gurobi, and CPLEX.

In multi-objective linear programming, to the best of our knowledge, only two recent papers [21, 22] belong to this second group. In [21], a matheuristic algorithm to approximate the non dominated frontier of bi-objective binary linear programs is proposed. The authors combine several exact/heuristic algorithms from both single and bi-objective linear programs with integer or continuous variables from the literature (the Chalmet method, the feasibility pump heuristic, a local search, and the Aneja and Nair method). The feasibility pump is designed for finding feasible solutions by using the weighted sum method. The local search is used for improving the solutions obtained by the feasibility pump heuristic, and the Chalmet method is used for finding solutions from different parts of the objective space. This matheuristic is extended in [22] for multi-objective mixed integer linear programs.

The algorithm we propose in this paper belongs to this second group. It aims to reuse the principles based on linear programming techniques structuring FP, in considering the multi-objective nature of the optimisation problem to solve (the single optimal solution is replaced by a set of efficient solutions). The components of the algorithm have been specified in order to have a low computational

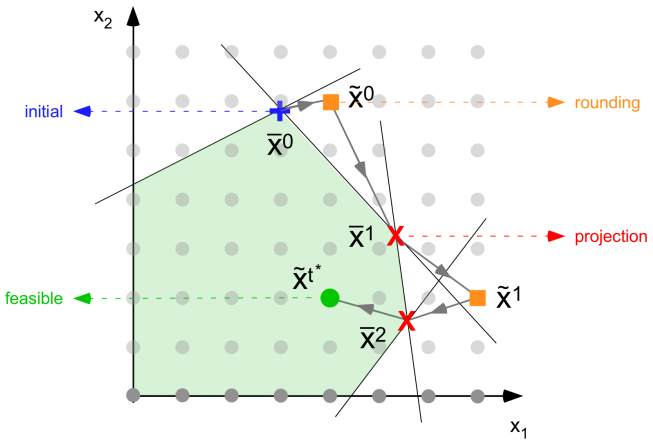

Figure 1: Illustration of the feasibility pump principle. $t \geq 0$ **is an iteration,** $\overline{x}^t$ **is a fractional solution,** $\tilde{x}^t$ **is an integer solution,** $\overline{x}^0$ **is an initial solution (e.g. here,** $\overline{x}^0 := \arg\min\{c^T x : x \in \overline{X}\}$ **i.e. a minimum-cost solution of the LP relaxation),** $\tilde{x}^{t^*}$ **is a feasible integer solution detected (when it exists) at iteration** t^*

complexity and for being scalable to more than two objectives. Due to the differences with FP, and to avoid any confusion with this latter, the proposed algorithm gets a specific name, which is « Gravity Machine » (for the association with the idea of flatting the image of feasible solutions identified along Y_N). For more detail see the report [11].

2 THE ALGORITHM

The ultimate goal targeted in our work is to tighten L and U over Y_N in order to prune as early as possible useless nodes in a branch-and-bound algorithm. The proposed algorithm is devoted to the upper bound set U. The desirable characteristic is to get a such U well representative of Y_N, and not necessary a close approximation of all $y \in Y_N$, which is a difference with [21, 22]. This section develops a canonical version of the algorithm which is numerically evaluated in section 3. Several potential variants and extensions are mentioned and will be investigated in the forthcoming versions.

2.1 Principle and components

The main principle adopted is, starting from a lower bound set L, to flatten the image of feasible solutions found on L, in order to collect a sample of points approximating Y_N. The main components structuring the algorithm are the following.

A lower bound set L. In this version, L is obtained by sampling with an ϵ-constraint method the polytope of the linear relaxation of the multi-objective optimisation problem. The two objectives (i.e. $p = 2$) are solved separately, aiming to deduce the range of values $[z^k_{min}, z^k_{max}], k = 1, 2$ on both objectives and thus the steps ϵ_1 and ϵ_2 for a given number of samples (parameter n_L). The image of solutions (symbol '+' on Figure 2) delimits the search area in the objective space.

A set of generators in the decision space. We call *generator* a starting LP-feasible solution (a priori unfeasible, i.e. not integer) used by the algorithm for trying to find a feasible solution. In this version, we consider as generators all solutions $\bar{x}^k, k = 1, \ldots, n_L$ such that their images $\bar{y}^k := z(\bar{x}^k)$ form the lower bound set L (ps: by abuse of language, $\bar{y}^k$ may sometimes also be designated by "generator", as shortcut for "the image of a generator" when no misunderstanding appears). Intuitively, a generator is valuable for the search of a feasible solution if its image in objective space is closer to Y_N. From a generator, the operations performed sequentially alternate rounding of a non-integer solution and projection of a solution on the polytope of the relaxed problem until detecting (i) a feasible solution or (ii) a cycling situation or (iii) a stopping condition (time limit or iteration limit given respectively by the parameters `maxTime` and `maxTrial`). Thereby a path composed of solution images takes shape from a generator (depicted on Figure 2 by $\bar{y}^{k,0} - \tilde{y}^{k,0} - \bar{y}^{k,1} - \tilde{y}^{k,t^*}$ where t^* is the final iteration along the path when a feasible solution is found -i.e. stopping condition (i) fulfilled-) and will be guided to Y_N by two cones defined hereafter.

A cone open to the search area in objective space. A cone is pointed on a generator $\bar{y}^k$, and is defined on base of the two adjacent images of generators $\bar{y}^{k-1}$ and $\bar{y}^{k+1}$ for the generator $\bar{y}^k$ (see Figure 2). The rays $]\bar{y}^{k-1}, \bar{y}^k]$ and $[\bar{y}^k, \bar{y}^{k+1}[$ define two hard constraints (blue plain lines on Figure 2) to respect by the rounded solutions (depicted by $\tilde{y}^{k,0}$ and $\tilde{y}^{k,t^*}$ and respectively symbols ■ and •) issued for this generator. Two dummy points $\bar{y}^0$ and $\bar{y}^{n_L+1}$ are pushed in L for the needs of the cones pointed on the two extreme points $\bar{y}^1$ and $\bar{y}^{n_L}$.

A cone open to the lower bound set in objective space. A cone is pointed on a projected point (depicted by point $\bar{y}^{k,1}$ and the symbol × on Figure 2), and is defined on base of the two adjacent images of generators ($\bar{y}^{k-1}$ and $\bar{y}^{k+1}$ for the generator $\bar{y}^k$ for which this projected point is derived). The rays $]\bar{y}^{k-1}, \tilde{y}^k(1)]$ and $[\tilde{y}^k(1), \bar{y}^{k+1}[$ define two soft constraints (dashed lines on Figure 2) to be satisfied if possible by the rounded solutions (depicted by $\tilde{y}^{k,1}$ and $\tilde{y}^{k,t^*}$ respectively symbols ■ and •).

These components are assembled within Algorithm 1 which outline « Gravity Machine ». With a simple design, an immediate analogy can be made with the first version of feasibility pump in terms of degree of sophistication. As expected, these components may be instantiated on problems with more than two objectives, knowing however that the question of scalability is never trivial in general in multi-objective optimisation.

2.2 Outline of the algorithm

This section outlines of the algorithm, with their parameters and their main variables. Main functions are enumerated with a short comment.

Parameters:

- $\mathcal{D}$: an instance of $2-01LP$ to solve

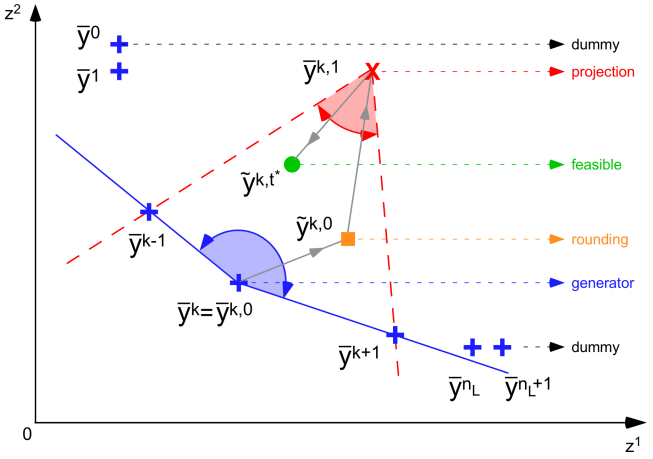

Figure 2: For a bi-objective minimisation problem, illustration of the two cones allowing to guide the search in the objective space from the generator $\bar{y}^k$. The points $\bar{y}^{k,0}$ to $\tilde{y}^{k,t^*}$ correspond respectively to the image of a generator solution (symbol '+'), a rounded solution (symbol '■'), a projected solution (symbol '×'), and a rounded and feasible solution (symbol '•'). From $\bar{y}^k$, the ordered list of points $\bar{y}^{k,0} - \tilde{y}^{k,0} - \bar{y}^{k,1} - \tilde{y}^{k,t^*}$ represents the path of visited solutions.

- n_L: the number of generators to compute
- `maxTrial`: the maximum trial to do per generator
- `maxTime`: the time limit allowed at each generator

Variables:

- $\bar{x}$: a solution where $x_i \in [0, 1],\ i = 1, n$
- $\tilde{x}$: a solution where $x_i \in \{0, 1\},\ i = 1, n$
- $F \subseteq X$: a set of feasible solutions
- H: a set historicizing the rounded solutions visited
- k: the k-th generator
- `trial`: number of attempts done on the k-th generator
- `feasible`: boolean set to true when a solution is feasible
- `cycle`: boolean set to true when a solution is revisited

computeGenerators The generators are computed by applying a double series of ϵ-constraint resolutions, one series per objective, with a step set up by the parameter n_L. Figure 3 shows a typical output returned with this generation procedure. Several others generation procedure could be also produce an upper bound set L such as applying the Aneja and Nair method with a well diversified set of weights or also in applying a parametric resolution method.

roundingSolution This rounding function transforms a fractional solution $\bar{y}^{k,t}$ into one binary solution $\tilde{y}^{k,t}$. This transformation is guided by the two cones defined on the objective space. The cone pointed on $\bar{y}^{k,0}$ avoids switching to zero undesirable variables x_j such that the resulting solution x is necessarily unfeasible. The cone pointed on $\tilde{y}^{k,t}, t > 0$, locates if possible the image of the resulting solution x in the neightborhood of $\bar{y}^{k,0}$. This action aims to generate a well distributed upper bound set U along Y_N. In the results

Data: $\mathcal{D}$, n_L, maxTrial, maxTime
Result: L, U
L, $F \leftarrow$ computeGenerators($\mathcal{D}$, n_L);
$H \leftarrow \emptyset$;
forall $z(\bar{x}^k) \in L, \bar{x}^k \notin F$ **do** // each k-th unfeasible gen.
  timeStart $\leftarrow$ time(); trial $\leftarrow$ 0;
  timeout $\leftarrow$ false; feasible $\leftarrow$ false;
  $\tilde{x}$, H, cycle $\leftarrow$ roundingSolution($\bar{x}^k$, k, H);
  while $\neg feasible$ && $\neg timeout$ **do**
    trial $\leftarrow$ trial + 1 ;
    $\bar{x}$, F, feasible $\leftarrow$ projectSolution($\tilde{x}$);
    if $\neg feasible$ **then**
      $\tilde{x}$, H, cycle $\leftarrow$ roundingSolution($\bar{x}$, k, H);
      if *cycle* **then**
        $\tilde{x} \leftarrow$ perturbSolution($\tilde{x}$);
      end
    end
    timeout $\leftarrow$ (time()-timeStart $\geq$ maxTime) || (trial = maxTrial)
  end
end
$U \leftarrow$ extractNondominatedPoints(F);

Algorithm 1: Outline of « Gravity Machine »

reported in this paper, each fractional variable to set to 0 or 1 is considered alone. One variant experimented with success considers the fractional variables 2 per 2. This variant enlarges slightly the neighborhood and thus enables the algorithm to visit advantageously more configurations.
One alternative is to replace the cone pointed on $\tilde{y}^{k,t}, t > 0$, by a projection guided by $\bar{y}^{k,0}$. This variant has been experimented but it is not reported in this paper.

projectSolution. Similar to the projection phase of feasibility pump, this function find through linear programming the point $\bar{x} \in \bar{X}$ which is as close as possible to the current binary solution $\tilde{x}$. If

$$\Delta(\bar{x}, \tilde{x}) = \sum_{j=1}^{n} \mid \bar{x}_j - \tilde{x}_j \mid = 0$$

then $\bar{x} = \tilde{x}$ is a 01 feasible solution, and we are done. As proposed in [9], this step is achieved in solving the following linear programing problem (given here coded in julia and using JuMP, an algebraic modeling language):

```
function delta(A,xInt)

  nbctr = size(A,1); nbvar = size(A,2)
  xInt0, xInt1 = split(xInt)

  m = Model(GLPK.Optimizer)
  @variable(m, 0.0 <= x[1:length(xInt)] <= 1.0)
  @objective(m, Min, sum(x[i] for i in xInt0)
                       + sum((1-x[i]) for i in xInt1))
  @constraint(m, [i=1:nbctr],
            (sum((A[i,j]*x[j]) for j in 1:nbvar)) == 1)
  optimize!(m)
  return objective_value(m), value.(x)
end
```

JuMP model 1: the projection model

In this model, `A` is the matrix of coefficients for the constraints, `xInt` corresponds to $\tilde{x}$, `split` is a function identifying the values 0 and 1 into $\tilde{x}$, `@objective` defines the objective function to minimise, `@constraint` defines the systems $Ax = 1$, `optimize!` invokes the MIP solver (here GLPK) on the model `m` stated, and the function returns the optimal values of $\overline{x} := \arg\min\{\Delta(x, \tilde{x}), x \in \overline{X}\}$.

perturbSolution. For a given iteration of the algorithm, $\tilde{x}$ is the current rounded solution and H records the previous rounded solution obtained. When $\tilde{x} \in H$, the two successive rounded solutions are identical which generates a 1-cycle. In that case a perturbation is applied.
The perturbation principle follows the principle introduced in the first version of feasibility pump. The differences concern the two cones. The value

$$\texttt{nbVar} \leftarrow \text{random}[\tfrac{T}{2}, \tfrac{3T}{4}] \qquad \text{with } T = 30$$

is computed and the `nbVar` first variables of $\bar{x}_i$ closest to 0.5 are considered as follows; if $\tilde{x}_i = 0$ (resp. $\tilde{x}_i = 1$) then flip $\tilde{x}_i = 1$ (resp. $\tilde{x}_i = 0$) if and only if the image of the solution belongs to the two cones.

extractNondominatedPoints At the end, the algorithm has generated at most n_L feasible solutions (and in the worst case, none feasible solution) gathered in the set F. This last step identifies the non-dominated points in F giving the upper bound set U. This is achieved in applying the Kung algorithm [17] on F. In this configuration ($p = 2$, set of points F given a priori and static) the complexity of this algorithm is in $O(n \log n)$ which makes it competitive in comparizon with other (brute-force or sophisticated) algorithms.

3 NUMERICAL EXPERIMENTS

The algorithm is coded in Julia language (version 1.6) [1, 14], uses the algebraic modeling language JuMP (version 0.21.8) [2, 15, 19] for representing and solving the optimisation problems. GLPK (version 4.65) [12] is selected as MIP solver in our experiments.

The bi-objective set partitioning problem (2-SPA) experimented is a particular (p-01LP) problem belonging to the class of MultiObjective Combinatorial Optimization problems (MOCO) [4]. Nevertheless, the specific structure of the SPA is not exploited. The 2-SPA can be formulated as follows:

$$\begin{array}{rcl} \min z(x) & = & Cx \\ \text{subject to } Ax & = & 1 \\ x & \in & \{0,1\}^n \end{array}$$

where $C \in \mathbb{Z}^{p \times n}$, and $A \in \{0,1\}^{m \times n}$.

The sets of non-dominated points Y_N of 2-SPA instances are obtained with the vOptSolver [10, 24], in particular with the package `vOptGeneric.jl` where the solving method selected is the ϵ-constraint method [13] with a step ϵ set to 1.

3.1 Illustration step by step

This illustration is based ont the dataset sppnw11 which contains 39 constraints and 8820 variables. The algorithm runs with:

- n_L= 10 generators,
- maxTrial= 5 trials per generator,
- maxTime= ∞ (no timeout activated).

The ranges Δ^1 and Δ^2 on objective 1 and 2 derived from the lower bound set L are:

z^1_{min}=116254.50 $\leftrightarrow$ z^1_{max}=134099.22 (Δ^1=17844.72)
z^2_{min}=38404.22 $\leftrightarrow$ z^2_{max}=70704.00 (Δ^2=32299.78)

Figures 3 to 6 depicte the activity of the algorithm at the end of its major steps. Figures 6 to 7 report the upper bound set U obtained for respectively the values n_L set to 10 and 30.

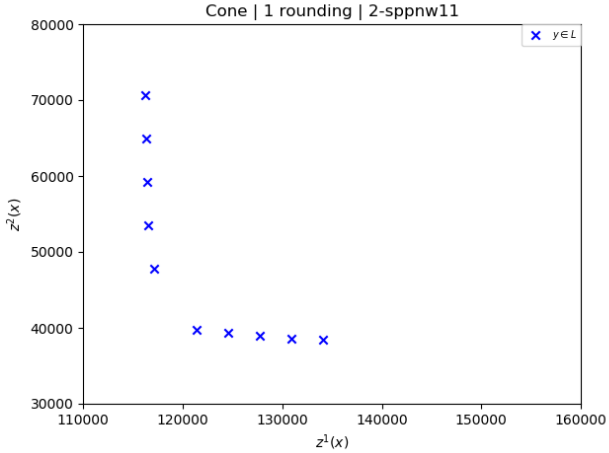

Figure 3: Lower bound set L computed.

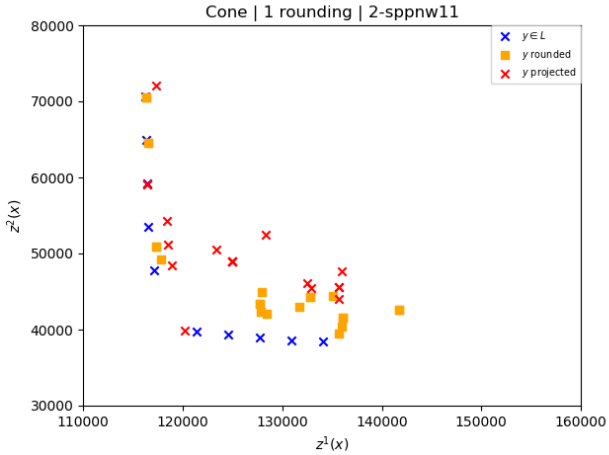

Figure 4: Rounded and projected points computed.

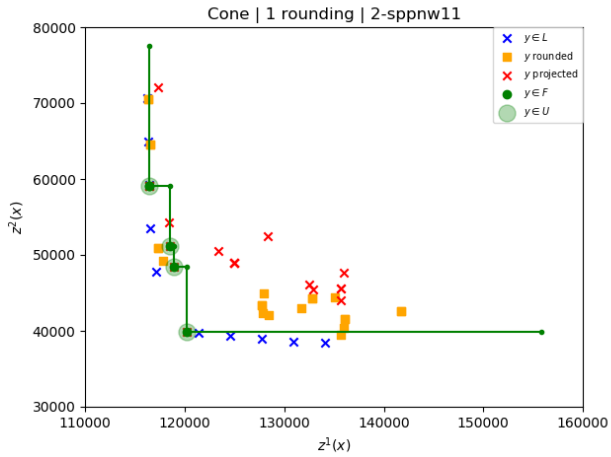

Figure 5: Feasible points computed and Upper bound set U obtained.

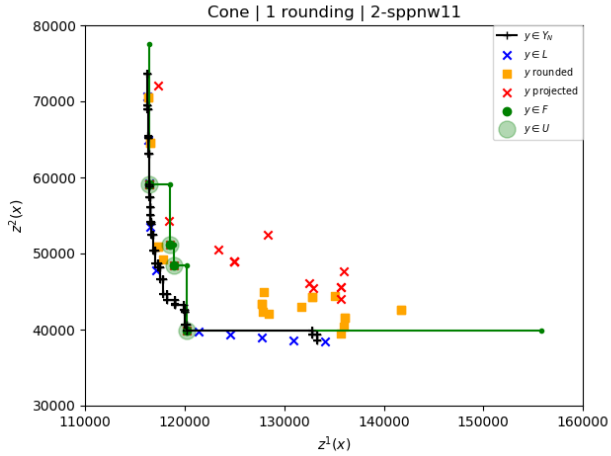

Figure 6: Comparizon of the Upper bound set U obtained and non-dominated points Y_N.

3.2 Numerical instances

Table 1 reports the dataset used for conducting the numerical experiments. The 44 instances come from the OR-library [20]. A second objective as been generated. The coefficients of the second objective is a random permutation of the coefficients of the first objective. In such a way, the numerical characteristics of both objectives are identical. This dataset will be uploaded on the vOptLib repository soon (https://github.com/vOptSolver/vOptLib).

3.3 Quality measure

To evaluate the quality of a U, the exclusive hypervolume contribution of solutions is computed (see Figure 8) and gives

$$r = \frac{A_U}{A_{Y_N}}$$

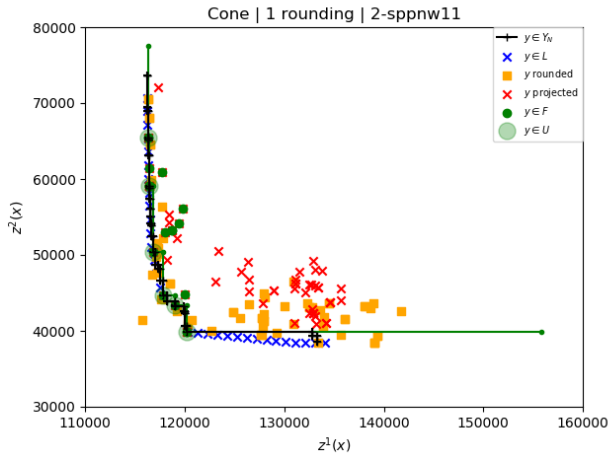

Figure 7: Run with 30 generators.

where

- A_U is the area covered by U
- A_{Y_N} is the area covered by Y_N
- the reference point is provided by the two extreme points of Y_N plus 1 on each extremal value.

Higher is the value of the indicator r, better is the upper bound set U.

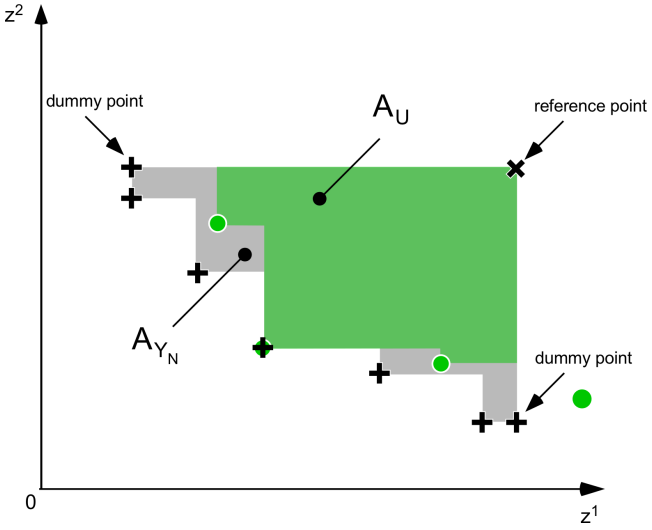

Figure 8: For a bi-objective minimisation problem, illustration of the quality measure. (symbol '+', $y \in Y_N$; symbol '•', $y \in U$; symbol '×', the reference point)

3.4 Numerical results

Experiments have been performed on a computer equipped with an Intel(R) Core(TM) i7-10870H processor running at 2,20 GHz and with 16 Go of RAM DDR4. The first experiment is done with $n_L = 10$, while the second is done with $n_L = 30$. The two other parameters are set to: `maxTrial=5` and `maxTime=`∞.

Table 2 report a first wave of results collected during the numerical experiments. For the 44 instances,

- $\#U$ shows the number of solutions detected and pushed into the upper bound set U,
- the CPUt in seconds consumed by the algorithm,
- r is the measure of the quality of U

For only one instance (`sppaa05`), the algorithm fails to return an upper bound set, when $n_L = 10$. For only two instances (`sppnw17` and `sppnw21`), the quality measure is lower with $n_L = 30$ than with $n_L = 10$. For 25 instances (57%), the quality indicator is greater of equal to 50% when $n_L = 10$. The situation increases to 34 instances (77%) when $n_L = 30$. Even if the current implementation of the proposed algorithm may be polished, the CPUt appears competitive especially for instances where computing Y_N is expensive.

4 CONCLUSION AND DISCUSSION

« Gravity Machine » is a primal heuristics inspired by « Feasibility Pump » to compute an upper bound set U for multi-objective 0-1 linear optimisation problems. The aims of the algorithm is to be competitive in quality and in CPUt in order to embed it into a multi-objective branch and bound algorithm. A very first version is presented, evaluated using instances of set partitioning problem and analyzed in comparison with the non-dominated points for all instances. The extended version of this paper will go further in the experimentation, as well in the details allowing to reproduce exactly the algorithm discussed.

This is an ingoing work and already the results are clearly encouraging. Many variants have been specified and several of them have been already experimented. In the mid term, the variants will be presented and analyzed, as well the experimentation on others optimisation problems.

REFERENCES

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671 arXiv:https://doi.org/10.1137/141000671

[2] Iain Dunning, Joey Huchette, and Miles Lubin. 2017. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Rev.* 59, 2 (2017), 295–320.

[3] Matthias Ehrgott. 2005. *Multicriteria Optimization.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[4] Matthias Ehrgott and Xavier Gandibleux. 2000. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum* 22, 4 (2000), 425–460.

[5] Matthias Ehrgott and Xavier Gandibleux. 2001. Bounds and Bound Sets for Biobjective Combinatorial Optimization Problems. In *Multiple Criteria Decision Making in the New Millennium (Lecture Notes in Economics and Mathematical Systems, Vol. 507)*, Murat Koksalan and Stan Zionts (Eds.). Springer, 241–253. 15th International conference in Multiple Criteria Decision-Making Proceedings.

[6] Matthias Ehrgott and Xavier Gandibleux. 2004. Approximative solution methods for multiobjective combinatorial optimization. *Top* 12, 1 (2004), 1–63. https://doi.org/10.1007/BF02578918

[7] Matthias Ehrgott and Xavier Gandibleux. 2007. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research* 34 (2007), 2674–2694.

[8] Michael Emmerich and André Deutz. 2018. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing* 17 (2018), 585–609. https://doi.org/10.1007/s11047-018-9685-y

[9] Matteo Fischetti, Fred Glover, and Andrea Lodi. 2005. The feasibility pump. *Mathematical Programming* 104, 1 (01 Sep 2005), 91–104. https://doi.org/10.1007/s10107-004-0570-3

[10] Xavier Gandibleux, Gauthier Soleilhac, Anthony Przybylski, and Stefan Ruzika. 2017. vOptSolver: an open source software environment for multiobjective

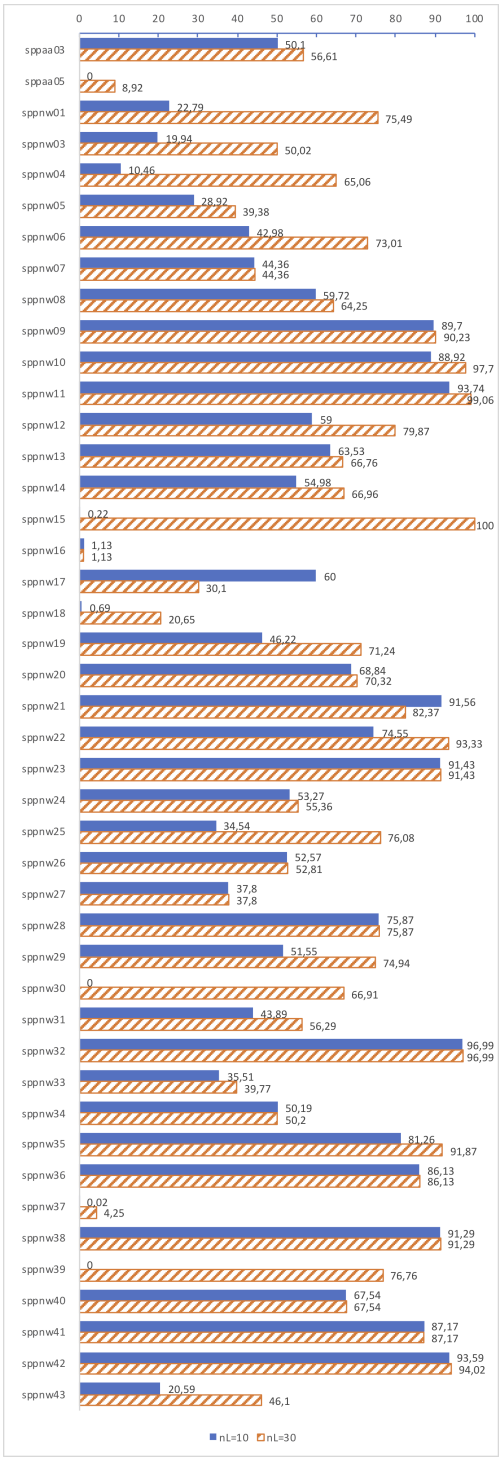

Figure 9: Plot of the quality measure for all instances when $n_L = 10$ and $n_L = 30$ (visual representation of the values reported in Table 2)

.

mathematical optimization. IFORS2017: 21st Conference of the International Federation of Operational Research Societies. July 17-21, 2017. Quebec City (Canada).

[11] Guillaume Gasnier. 2021. Heuristiques primales multi-objectifs. Mémoire de "Travail d'étude et de recherche", Université de Nantes.

[12] GLPK. [n.d.]. GNU Linear Programming Kit. https://www.gnu.org/software/glpk/. Accessed: 2021-06-05.

[13] Yacov V. Haimes, Leon S. Lasdon, and David A. Wismer. 1971. On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1, 3 (1971), 296–297.

[14] Julia. [n.d.]. The Julia Language. https://julialang.org/. Accessed: 2021-06-05.

[15] JuMP. [n.d.]. A modeling language for mathematical optimization embedded in Julia. https://github.com/jump-dev/JuMP.jl. Accessed: 2021-06-05.

[16] Abdullah Konak, David W. Coit, and Alice E. Smith. 2006. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety* 91, 9 (2006), 992–1007. https://doi.org/10.1016/j.ress.2005.11.018 Special Issue - Genetic Algorithms and Reliability.

[17] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (Oct. 1975), 469–476. https://doi.org/10.1145/321906.321910

[18] Qi Liu, Xiaofeng Li, Haitao Liu, and Zhaoxia Guo. 2020. Multi-objective metaheuristics for discrete optimization problems: A review of the state-of-the-art. *Applied Soft Computing* 93 (2020), 106382. https://doi.org/10.1016/j.asoc.2020.106382

[19] Miles Lubin and Iain Dunning. 2015. Computing in Operations Research Using Julia. *INFORMS Journal on Computing* 27, 2 (2015), 238–248.

[20] OR-library. [n.d.]. A collection of test data sets for a variety of OR problems. http://people.brunel.ac.uk/ mastjjb/jeb/orlib/sppinfo.html. Accessed: 2021-06-05.

[21] Aritra Pal and Hadi Charkhgard. 2019. A Feasibility Pump and Local Search Based Heuristic for Bi-Objective Pure Integer Linear Programming. *INFORMS Journal on Computing* 31, 1 (2019), 115–133. https://doi.org/10.1287/ijoc.2018.0814

[22] Aritra Pal and Hadi Charkhgard. 2019. FPBH: A feasibility pump based heuristic for multi-objective mixed integer linear programming. *Computers & Operations Research* 112 (2019), 104760. https://doi.org/10.1016/j.cor.2019.07.018

[23] Anthony Przybylski and Xavier Gandibleux. 2017. Multi-objective branch and bound. *European Journal of Operational Research* 260, 3 (2017), 856–872. https://doi.org/10.1016/j.ejor.2017.01.032

[24] vOptSolver. [n.d.]. Solver of multiobjective linear optimization problems. https://github.com/vOptSolver Accessed: 2021-06-05.

Table 1: Numerical instances, with their characteristics (number of constraints, number of variables), and the exact solutions (Y_N, CPUtime consumed)

Instances	#Constraints	#Variables	$\#Y_N$	CPUt (s)
sppaa03	825	8627	265	1814.02
sppaa05	801	8308	310	2564.50
sppnw01	135	51975	421	6031.67
sppnw03	59	43749	47	365.49
sppnw04	36	87482	26	584.73
sppnw05	71	288507	73	4604.59
sppnw06	50	6774	20	29.32
sppnw07	36	5172	20	16.40
sppnw08	24	434	20	0.29
sppnw09	40	3103	22	5.99
sppnw10	24	853	13	0.20
sppnw11	39	8820	28	12.88
sppnw12	27	626	43	1.61
sppnw13	51	16043	125	503.94
sppnw14	73	123409	103	19082.31
sppnw15	31	467	2	0.15
sppnw16	139	148633	7	519.62
sppnw17	61	118607	58	26875.33
sppnw18	124	10757	23	27.15
sppnw19	40	2879	20	5.19
sppnw20	22	685	9	0.71
sppnw21	25	577	10	0.46
sppnw22	23	619	19	1.21
sppnw23	19	711	6	0.16
sppnw24	19	1366	21	1.77
sppnw25	20	1217	17	1.07
sppnw26	23	771	19	0.50
sppnw27	22	1355	12	0.49
sppnw28	18	1210	6	0.28
sppnw29	18	2540	12	2.62
sppnw30	26	2653	8	1.82
sppnw31	26	2662	16	3.59
sppnw32	19	294	8	0.08
sppnw33	23	3068	10	2.79
sppnw34	20	899	14	0.70
sppnw35	23	1709	14	1.20
sppnw36	20	1783	10	3.66
sppnw37	19	770	10	1.29
sppnw38	23	1220	8	1.15
sppnw39	25	677	12	0.81
sppnw40	19	404	10	0.25
sppnw41	17	197	11	5.19
sppnw42	23	1079	10	0.77
sppnw43	18	1072	21	7.03

Table 2: Numerical results for each instances (number of points within the upper bound set U and CPUtime consumed)

Instances	$\#U$	CPUt (s)	r (%)	$\#U$	CPUt (s)	r (%)
sppaa03	3	85.56	50.1	4	433.92	56.61
sppaa05	0	73.55	0.0	2	398.04	8.92
sppnw01	3	37.06	22.79	9	186.98	75.49
sppnw03	1	24.95	19.94	4	53.91	50.02
sppnw04	2	17.0	10.46	3	71.87	65.06
sppnw05	5	96.33	28.92	9	245.93	39.38
sppnw06	1	1.87	42.98	4	5.03	73.01
sppnw07	4	0.87	44.36	4	3.22	44.36
sppnw08	5	0.04	59.72	7	0.13	64.25
sppnw09	5	0.42	89.7	5	1.65	90.23
sppnw10	6	0.08	88.92	7	0.26	97.7
sppnw11	4	1.34	93.74	6	4.24	99.06
sppnw12	4	0.07	59.0	6	0.19	79.87
sppnw13	2	2.47	63.53	7	9.09	66.76
sppnw14	4	30.35	54.98	8	113.73	66.96
sppnw15	2	0.07	0.22	3	0.14	100.0
sppnw16	2	76.78	1.13	2	223.7	1.13
sppnw17	1	71.19	60.0	2	148.67	30.10
sppnw18	2	6.24	0.69	1	21.22	20.65
sppnw19	3	0.38	46.22	3	1.19	71.24
sppnw20	1	0.07	68.84	2	0.21	70.32
sppnw21	2	0.06	91.56	3	0.18	82.37
sppnw22	3	0.09	74.55	4	0.22	93.33
sppnw23	3	0.08	91.43	4	0.22	91.43
sppnw24	3	0.16	53.27	3	0.43	55.36
sppnw25	2	0.15	34.54	3	0.43	76.08
sppnw26	2	0.08	52.57	4	0.24	52.81
sppnw27	1	0.15	37.8	1	0.48	37.8
sppnw28	2	0.13	75.87	2	0.31	75.87
sppnw29	2	0.26	51.55	4	0.71	74.94
sppnw30	1	0.25	0.0	1	1.03	66.91
sppnw31	2	0.38	43.89	3	1.19	56.29
sppnw32	3	0.06	96.99	3	0.09	96.99
sppnw33	1	0.41	35.51	4	1.21	39.77
sppnw34	1	0.09	50.19	2	0.29	50.2
sppnw35	2	0.21	81.26	4	0.58	91.87
sppnw36	1	0.19	86.13	1	0.56	86.13
sppnw37	1	0.08	0.02	2	0.2	4.25
sppnw38	2	0.11	91.29	2	0.32	91.29
sppnw39	2	0.16	0.0	3	0.32	76.76
sppnw40	1	0.05	67.54	1	0.13	67.54
sppnw41	3	0.02	87.17	3	0.07	87.17
sppnw42	2	0.14	93.59	3	0.36	94.02
sppnw43	4	0.12	20.59	5	0.3	46.1
	n_L=10			n_L=30		