A combinatorial branch-and-bound algorithm for box search

Q. Louveaux, S. Mathieu*

Institut Montefiore (B28), University of Liège
Grande Traverse 10, B-4000
Liège, Belgium

Abstract

Considering a set of points in a multi-dimensional space with an associated real value for each point, we want to find the box with the maximum sum of the values of the included points. This problem has applications in data mining and can be formulated as a mixed-integer linear program. We propose a branch-and-bound algorithm where the bounding is obtained by combinatorial arguments instead of the traditional linear relaxation. Computational experiments show that this approach competes with current state of the art mixed-integer solvers. The algorithm proposed in this paper may be seen as a simple and dependence-free method to solve the box search problem.

Keywords: Combinatorial optimization, Branch-and-bound, Data mining, Mixed integer programming
2010 MSC: 90C27, 90C57, 90C11, 97R50

1. Introduction

Data mining [9] has been a popular field of research recently since it is possible to generate big databases. It consists in finding relevant information in a large set of data that is usually automatically generated. The most well-known example is when a company collects information about its customers in order to provide a better individualized service. Together with this new trend, companies also try to automatically collect as much data as possible. An important question is therefore to understand how to handle this new information correctly. In this paper, we are interested in a particular problem that arises in this context and that may be of interest for very complex industrial processes. In particular, we consider a process for which we automatically retrieve a number of variables that might influence a given output variable. For example, we may consider a line creating some products and the output variable could be an estimation of the quality of the product. It is clear that a large number of variables may influence the quality of the production and we assume that most of them can be retrieved along the way for each produced item. The question that we ask is to find a set of rules, i.e. intervals on the variables, for which the average output is maximized. This can be of interest in order to find a setup that works well in average. On the other hand, if we find a set of rules for which the average output is minimized, this could also potentially explain why a number of objects is of bad quality. Both questions are of interest in the context of understanding an industrial process and we try to address them in this paper.

*Corresponding author. Phone number: +32 4 366 4814
Email address: {q.louveaux, sebastien.mathieu}@ulg.ac.be (Q. Louveaux, S. Mathieu)

More specifically, we assume that each produced item i is represented by a D-dimensional vector x^i of the input variables for which we consider a normalized value in [0,1]. The output value of the item i is given by $c^i \in \mathbb{R}$ and might be either positive or negative. The problem addressed in this paper is to find a box, i.e. an interval $[l_t, u_t]$ for each dimension t, which maximizes the sum of the values c^i of each point i included in the box. We name this problem the *box search*. The purpose of a box is its simplicity and robustness.

A way to handle data mining is by using machine learning strategies. In [10], Friedman et al. introduce the heuristic PRIM (Patient Rule Induction Method) for bump-hunting in high-dimensional data. The procedure starts with a box including all points. At each iteration, PRIM peels α points of the box by restricting the box on one dimension. The procedure stops when the number of points included in the box drops below a fraction β of the total number of points. Several improvements of this technique have been proposed as PRIM2 [7] or f-PRIM [5]. Other alternatives in this area are suggested like subgroup discovery [14] or a genetic algorithm [18].

A common drawback of these approaches is that they give no guarantee on the quality of the solution. One may however require finding the best box and provide a certificate of optimality. Unfortunately, this kind of problem is known to be NP-Hard. It can be trivially solved in $M^{O(D)}$ operations where M is the number of points and D their dimension [12]. Eckstein et al. introduce in [8] the *maximum box problem* where we seek a box maximizing the weighted sum over a set of points with a positive objective value while not intersecting a set of given points. They prove that the *maximum box problem* is NP-Hard in the general case and polynomial if we fix the dimension. The maximum box problem trivially reduces to the box search which implies that it is NP-hard as well. This polynomial complexity is particularly interesting for the two-dimensional case where algorithms of low complexity can be implemented [16, 6].

A classic approach to solve NP-Hard problems to global optimality is the branch-and-bound algorithm [13]. It has been applied successfully in [11] to find logical patterns where the branching decision is made on the inclusion or the exclusion of one logical element. A combinatorial branch-and-bound algorithm has been proposed to solve the maximum box problem where every child is created by dividing the space along each dimension to exclude a point which must not be intersected by the box [8]. However the algorithm proposed in [8] creates 2^D branches at each node in the worst case which may be prohibitive for high dimension. The main difference between the box search and [8] is that we do not restrict ourselves to homogeneous boxes of positive points. Whereas in [8], the goal is to find the box with a maximum number of points with a positive value without any point from an excluded set, we find the box with the maximum weighted sum of points with any value, positive or negative.

We show that the computation of the box with the maximum sum can be formulated as a mixed-integer linear program. We propose a combinatorial branch-and-bound approach to tackle the problem. The branching decision is not made on the variables of the linear model but on a decision of including or excluding a point from a candidate box. The bounding is obtained by combinatorial arguments instead of the traditional linear relaxation. We investigate common branching strategies such as strong branching [3, 15] and reliability branching [2] as well as one problem specific strategy. Computational experiments show that this approach compete with state of the art mixed-integer linear programming (MILP) softwares for this particular problem.

The outline is as follows. Section 2 proposes a MILP formulation of the box search problem. We present our combinatorial branch-and-bound algorithm in Section 3. We compare the performance of the proposed algorithm with a standard MILP software in

Section 4 and discuss the performance of the proposed algorithm.

2. Integer programming models

In this section we formulate the box search problem as an integer program. We propose two formulations for the problem. In the first formulation, we propose to use the bounds of the box as continuous variables of the problem. The second formulation is purely binary.

We explore a D-dimensional space with M points normalized with linear scaling transform [17]. The coordinates of a point i are denoted $\mathbf{x}^i$ with $\mathbf{x}^i \in [0,1]^D$. The value of a point i is $c^i \in \mathbb{R}$. Those points are partitioned in two sets: the set of positive points $\mathcal{P} = \{i \in \{1,...,M\} | c^i > 0\}$ and the set of negative points $\mathcal{N} = \{i \in \{1,...,M\} | c^i < 0\}$. Points such that $c^i = 0$ can be ignored.

To improve the robustness of the models, we apply the following transformation to the coordinates of the points. For each dimension $t \in \{0,...,D\}$, the points are sorted with respect to their coordinate $\{x_t^i, \forall i \in \{1,...,M\}\}$. Due to some points having the same coordinates on dimension t, we observe in this sorted list K_t distinct values, $K_t \leq M$. We replace the coordinate of the point i, x_t^i by its order in the sorted list of points divided by the number of elements K_t. The index of i in this sorted list is denoted r_t^i.

2.1. Continuous box bounds

In this formulation, a box is defined by its two opposite corners: $\mathbf{l} \in [0,1]^D$ and $\mathbf{u} \in [0,1]^D$. We introduce one binary variable z^i per point where $z^i = 1$ if the point i is included in the box and 0 otherwise. The box search can be formulated as the following optimization problem:

$$\max f = \sum_{i=1}^{M} c^i z^i \tag{1}$$

subject to:
$\forall t \in \{1,...,D\}$:

$$l_t \leq u_t \tag{2}$$

$\forall i \in \mathcal{P}$, $t \in \{1,...,D\}$:

$$l_t \leq x_t^i z^i + (1 - z^i) \tag{3}$$

$$x_t^i z^i \leq u_t \tag{4}$$

$\forall i \in \mathcal{N}$, $t \in \{1,...,D\}$:

$$(v_t^i + \eta_t) \geq (x_t^i - l_t) + \eta_t \tag{5}$$

$$(w_t^i + \eta_t) \geq (u_t - x_t^i) + \eta_t \tag{6}$$

$\forall i \in \mathcal{N}$:

$$z^i \geq \sum_{t=1}^{D} (v_t^i + w_t^i) - 2D + 1 \tag{7}$$

with $\forall i \in \{1,...,M\} : z^i, v_d^i, w_d^i \in \{0,1\}$ and $\forall t \in \{1,...,D\} : l_t, u_t \in [0,1]$ and $\eta_t = \frac{1}{2K_t}$.

Aside from (1) which computes the objective function of the box and the obvious constraint (2), the formulation can be divided in two main parts: (3)-(4) determine whether positive points are in the box or not and (5)-(7) determine whether negative points are in the box.

We first explain (3)-(4). Observe that if $z^i = 1$, the inequalities imply that $l_t \leq x_t^i \leq u_t$. On the other hand if $z^i = 0$, the constraints are always trivially satisfied. This set of constraints works only for the positive points. Indeed if the coefficient of the objective is negative, this set of constraints would imply that $z^i = 0$ even if all coordinates are in the box. It is therefore needed to work differently for points with values $c^i < 0$.

We now explain (5)-(7). Inequality (5) defines an auxiliary binary variable v_t^i for every negative point i in dimension t which is equal to 1 if $x_t^i \geq l_t$. Similarly, the auxiliary variable w_t^i is equal to 1 if $x_t^i \leq u_t$. A negative point i is included in the box if all corresponding auxiliary variables are equal to one, i.e. $\forall t \in \{1, ..., D\}, l_t \leq x_t^i \leq u_t$ or $\forall t \in \{1, ..., D\}, v_t^i = 1$ and $w_t^i = 1$. This condition is given by (7).

Formulation (1)-(7) is compact with its $M + 2D + 2|\mathcal{N}|D$ variables and $(2\mathcal{P} + 1)D + \mathcal{N}(1+2D)$ constraints. Unfortunately, this formulation is weak. If we relax the integrality constraints, we are likely to obtain as an optimal solution $z^i = 1\ \forall i \in \mathcal{P}$ and $z^i = 0\ \forall i \in \mathcal{N}$, using fractional values for v_t^i and w_t^i. The 1 in equation (7) is easily compensated by the relaxed auxiliary variables v_t^i and w_t^i which deactivate the constraint. In a linear programming based branch-and-bound algorithm, we would branch on variables v_t^i and w_t^i, the other variables already being integral, until one z^i is constrained to be strictly greater than zero. The objective value drops only when we finally branch on these variables z^i.

To avoid as much as possible numerical errors, the final box should be built from the variables z^i. The box is computed in each dimension t by its upper and lower bound, l_t and u_t such that $l_t = \min_{i \in \{1,...,M\}: z^i = 1} x_t^i$ and $u_t = \max_{i \in \{1,...,M\}: z^i = 1} x_t^i$.

2.2. Pure discrete formulation

This formulation introduces for each dimension t, a set of binary variables $\mathbf{y}_t \in \{0, 1\}^{K_t}$, y_t^m equals one if value m is part of the projection of the candidate box onto coordinate t. This formulation uses as data r_t^i, the index of point i in the sorted list of points in dimension t. The alternative MILP formulation is the following:

$$\max f = \sum_{i=1}^{M} c^i z^i \tag{8}$$

subject to:
$\forall i \in \mathcal{P}, t \in \{1, ..., D\}$:

$$z^i \leq y_t^{r_t^i} \tag{9}$$

$\forall i \in \mathcal{N}$:

$$z^i + \sum_{t=1}^{D} (1 - y_t^{r_t^i}) \geq 1 \tag{10}$$

$\forall t \in \{1, ..., D\}, k \in \{1, ..., K_t\}$:

$$y_t^k = p_t^k - q_t^k \tag{11}$$

$\forall t \in \{1, ..., D\}, k \in \{1, ..., K_t - 1\}$:

$$p_t^k \leq p_t^{k+1} \tag{12}$$

$$q_t^k \leq q_t^{k+1} \tag{13}$$

Inequality (9) expresses that a positive point may be selected if, for all dimension, its coordinates lies in the projection of the box on this dimension. (10) defines that a

negative point may be excluded if at least one of its coordinates is outside of the box's projection. The variables y_t^k must represent the projection of a box on the dimension t. This consecutivity property is modeled by (11)-(13) with auxiliary variables p_t^k and q_t^k. Once again, the solution box can be built from the variables z^i.

If we take $K_t = M \ \forall t \in \{1, ..., M\}$, formulation (8)-(13) introduces $(3D+1)M$ binary variables and $|\mathcal{P}|D+|\mathcal{N}|+3DM-2D$ constraints which is of the same order of magnitude as the mixed-integer formulation. Section 4 shows a comparison of the performance of both formulations.

3. Combinatorial branch-and-bound

We now propose a specific algorithm for this problem based on a branch-and-bound approach. We do not branch on variables. The branching decision is made on the inclusion or exclusion of each point from the box. We first introduce some notations.

Definition 1. *Let $\{1, ..., M\}$ be the set of points in the database. Each node of the branch-and-bound is denoted by $n(\mathcal{I}, \mathcal{E})$ indicating the status of fixing during the branch-and-bound. $\mathcal{I} \subseteq \{1, .., M\}$ is the set of points fixed or inferred to belong to the box and $\mathcal{E} \subseteq \{1, .., M\}$ is the set of points fixed or inferred to be out of the box. Note that $\mathcal{I} \cap \mathcal{E} = \emptyset$. The set of unfixed points is denoted $\mathcal{U}$ and is defined as $\mathcal{U} = \{1, ..., M\} \setminus (\mathcal{I} \cup \mathcal{E})$.*

In the root node, every point from the database is *unfixed*: $\mathcal{I} = \emptyset$, $\mathcal{E} = \emptyset$ and $\mathcal{U} = \{1, ..., M\}$. For each branching decision, we select one unfixed point $i \in \mathcal{U}$ and create two branches: $n(\mathcal{I} \cup \{i\}, \mathcal{E})$ with i included and $n(\mathcal{I}, \mathcal{E} \cup \{i\})$ with i excluded.

Observe that it is possible to base the branching process either on the list of positive points or on the whole set of points. In the first case, the negative points are included only if the included positive points force the negative points to be included. The algorithm explained in this article can be applied to both cases. For the sake of conciseness, we explain the case where we branch on the whole set of points. The computational results will later show a comparison of the performances of both choices.

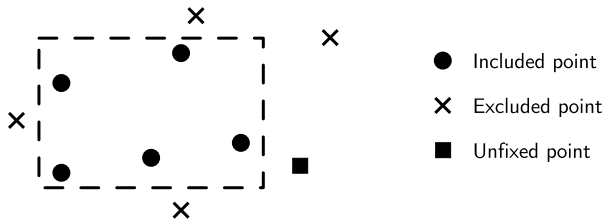


Figure 1: Representation of the operational box for a node in 2 dimensions.

A fixing $n(\mathcal{I}, \mathcal{E})$ defines a box $B(\mathcal{I})$ including all the fixed included points as illustrated in Figure 1 for a two dimensional database. This operational box $B(\mathcal{I})$ gives a primal solution $\underline{f}(\mathcal{I})$ and therefore a lower bound on the optimal value.

Definition 2. *Consider an arbitrary node $n(\mathcal{I}, \mathcal{E})$ of the branch-and-bound search tree with $\mathcal{I} \neq \emptyset$. We define the* operational box $B(\mathcal{I})$ *as the box*

$$B(\mathcal{I}) = \left\{\mathbf{x} \in [0,1]^D : x_t \in [l_t, u_t] \ \forall t \in \{1, ..., D\}\right\} \tag{14}$$

where $l_t = \min_{i \in \mathcal{I}} x_t^i$ and $u_t = \max_{i \in \mathcal{I}} x_t^i$. The projection of $B(\mathcal{I})$ on the dimension t is denoted $B_t(\mathcal{I})$ and corresponds to the range $B_t(\mathcal{I}) = [l_t, u_t]$.

Observation 1. *Consider a node* $n(\mathcal{I}, \mathcal{E})$*, a* primal solution $\underline{f}(\mathcal{I})$ *is given by*

$$\underline{f}(\mathcal{I}) = \sum_{i:\mathbf{x}^i \in B(\mathcal{I})} c^i \tag{15}$$

We can also infer an upper bound $\overline{f}(\mathcal{I}, \mathcal{E})$ on the best possible box considering the fixing information. This is explained in detail in Section 3.2. The values $\underline{f}(\mathcal{I})$ and $\overline{f}(\mathcal{I}, \mathcal{E})$ allow us to implement a branch-and-bound algorithm. This combinatorial branch-and-bound approach is described in Algorithm 1. In the very generic Algorithm 1, there are four important subprocedures that we need to explain in more details:

1. the inference process,
2. the computation of the primal solution $\underline{f}(\mathcal{I})$,
3. the computation of the upper bound $\overline{f}(\mathcal{I}, \mathcal{E})$,
4. the branching strategy i.e, the selection of the index $i \in \mathcal{U}$ to branch on.

Theses subprocedures are explained in details in Sections 3.1, 3.2 and 3.3.

Algorithm 1 Branch on points

Initialize the global lower bound $\underline{f}_g \leftarrow 0$, and the workpile $W \leftarrow \{n(\emptyset, \emptyset)\}$
while $W \neq \emptyset$ **do**
  Select one node $n(\mathcal{I}, \mathcal{E})$ from W and remove it from W.
  if $\overline{f}(\mathcal{I}, \mathcal{E}) \leq \underline{f}_g$ **then**
    continue
  end if
  Select a point $i \in \mathcal{U}$ to branch on
  $\mathcal{I}^+, \mathcal{E}^+ \leftarrow \texttt{inference}(\mathcal{I} \cup \{i\}, \mathcal{E})$
  Compute $\overline{f}(\mathcal{I}^+, \mathcal{E}^+)$
  $\underline{f}_g \leftarrow \max\{\underline{f}_g, \underline{f}(\mathcal{I}^+)\}$
  $W \leftarrow W \cup \{n(\mathcal{I}^+, \mathcal{E}^+)\}$
  $\mathcal{E}^- \leftarrow \texttt{inference}(\mathcal{I}, \mathcal{E} \cup \{i\})$
  Compute $\overline{f}(\mathcal{I}, \mathcal{E}^-)$
  $W \leftarrow W \cup \{n(\mathcal{I}, \mathcal{E}^-)\}$
end while

3.1. Inference of inclusion or exclusion of a point

In this section, we discuss how to exploit the information gathered by including or excluding a point i. We consider one node $n(\mathcal{I}, \mathcal{E})$, the corresponding operational box $B(\mathcal{I})$ and the primal solution $\underline{f}(\mathcal{I})$. In this section, we study the possibility to infer the fixing of some points in $\mathcal{U}$, the set of unfixed points.

First, we investigate the case where the algorithm excludes the point i to obtain the node $n(\mathcal{I}, \mathcal{E} \cup \{i\})$. As no new points are included, the primal solution $\underline{f}(\mathcal{I})$ is left unchanged. Excluding a point might fix some unfixed points as illustrated in Figure 2 where the last unfixed point can be excluded.

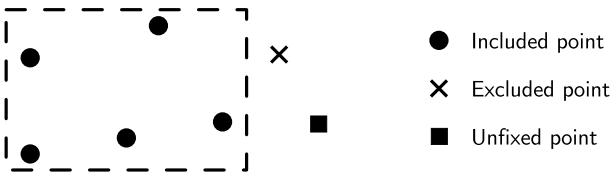


Figure 2: Considering a node, the last unfixed point must be excluded.

Observation 2. *Consider a fixing $n(\mathcal{I}, \mathcal{E})$, $\mathcal{I} \neq \emptyset$. An unfixed point $i \in \mathcal{U}$ is inferred to be out of the box if there exists an excluded point $j \in \mathcal{E}$ such that $j \in B(\mathcal{I} \cup \{i\})$.*

To check efficiently the inference of the exclusion, the following criteria can be used.

Lemma 1. *Consider a fixing $n(\mathcal{I}, \mathcal{E})$. An unfixed point $i \in \mathcal{U}$ is inferred to be out of the box by an excluded point $j \in \mathcal{E}$ if $\forall t \in \{1, ...D\}$,*

if $l_t > x_t^j$ ***then***
  $x_t^i \leq x_t^j$
else if $u_t < x_t^j$ ***then***
  $x_t^i \geq x_t^j$
end if

where $[l_t, u_t]$ is the projection of $B(\mathcal{I})$ as in Definition 2.

Proof. From Observation 2, a point i is inferred to be excluded by a point $j \in \mathcal{E}$ if $j \in B(\mathcal{I} \cup \{i\})$. The point j must be excluded in at least one dimension for every box $B(\mathcal{I}') : \mathcal{I} \subseteq \mathcal{I}' \subseteq \mathcal{I} \cup \mathcal{U}$ and in particular for the box $\mathcal{I} \cup \{i\}$. For every dimension $t \in \{1, ..., D\}$, there are three cases to consider:

1. $x_t^j < l_t$: If $x_t^i \geq x_t^j$, a box excluding the point j by the dimension t and including the points $\mathcal{I} \cup \{i\}$ can exist.

2. $x_t^j > u_t$: If $x_t^i \leq x_t^j$, a box excluding the point j by the dimension t and including the points $\mathcal{I} \cup \{i\}$ can exist.

3. $x_t^j \in [l_t, u_t]$: In this case, j can not be excluded by the dimension t by any box $B(\mathcal{I}') : \mathcal{I} \subseteq \mathcal{I}' \subseteq \mathcal{I} \cup \mathcal{U}$ and no condition on the coordinate x_t^i is needed.

□

Algorithm 2 summarizes the steps to check efficiently if a point i is inferred to be excluded if the point j is excluded. Note that the complexity of this algorithm is a linear function of the dimension D.

Now, we investigate the consequences of including a point i to obtain the node $n(\mathcal{I} \cup \{i\}, \mathcal{E})$ and the corresponding operational box $B(\mathcal{I} \cup \{i\})$. As the operational box changes, the primal solution $\underline{f}(\mathcal{I} \cup \{i\})$ needs to be computed. One way to compute the primal solution $\underline{f}(\mathcal{I} \cup \{i\})$ is for each point to check if it is included in $B(\mathcal{I} \cup \{i\})$ directly following Definition 1. This conducts to a complexity of order DM. The computation can be carried out more efficiently by retaining for each dimension t and each point $j \in \mathcal{U}$ if $x_t^j \in B_t(\mathcal{I} \cup \{i\}) \setminus B_t(\mathcal{I})$. Note that $B_t(\mathcal{I} \cup \{i\}) \setminus B_t(\mathcal{I})$ is an interval. Computing the score this way has still a worst case complexity of DM but in practice only a few points need to be checked for each dimension.

In the same time, other points can be inferred to be in the box as shown in Figure 3 where the last unfixed point must be included.

Algorithm 2 Inference of the exclusion of a point i by an excluded point j

for all $t = 1, ...D$ **do**
  if $l_t > x_t^j$ **then**
    if $x_t^i \geq x_t^j$ **then return** $i \notin \mathcal{E}$
  else if $u_t < x_t^j$ **then**
    if $x_t^i \leq x_t^j$ **then return** $i \notin \mathcal{E}$
  end if
end for
return $i \in \mathcal{E}$

Figure 3: Considering a node, the last unfixed point must be included.

Observation 3. *Consider a node $n(\mathcal{I}, \mathcal{E})$ and an unfixed point $j \in \mathcal{U}$. Point j is inferred to be in the box if $\mathbf{x}^j \in B(\mathcal{I})$.*

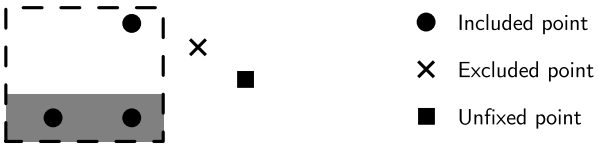


Figure 4: Including the point on the top implies that the last unfixed point must be excluded.

If a point is included, some other points may be inferred to be excluded as illustrated in Figure 4. The initial operational box is shown in gray. If we include the point on the top, the operational box becomes the one represented by the dashed strokes. In this case, the last unfixed point must be excluded. Therefore, if we include one new point, we should apply Algorithm 2 for every previously excluded point. Note that we can forget the inferred excluded points as stated by the next lemma.

Lemma 2. *Consider a node $n(\mathcal{I}, \mathcal{E})$ and a point $i \in \mathcal{E}$ inferred to be excluded by a point $j \in \mathcal{E}$. If a point k is inferred to be excluded by the point i then the point k is inferred to be excluded by the point j.*

Proof. The proof is based on Lemma 1. If $x_t^j < l_t$ then

- $x_t^j \geq x_t^i$ because point i is inferred to be excluded by the point j.
- $x_t^i \geq x_t^k$ because point k is inferred to be excluded by the point i.

Therefore, we have $l_t > x_t^j \geq x_t^i \geq x_t^k$ and point k is inferred to be excluded by the point j. The proof for the case where $x_t^j > u_t$ is similar. □

3.2. Finding an upper bound

The performance of branch-and-bound algorithms is highly dependent on the quality of the upper bound used. This section presents methods to obtain good upper bounds for this problem. We consider one node $n(\mathcal{I}, \mathcal{E})$ and we compute an upper bound $\overline{f}(\mathcal{I}, \mathcal{E})$.

Definition 3. *Consider a node $n(\mathcal{I}, \mathcal{E})$, we define the value of the best box that can be built in the subtree of root $n(\mathcal{I}, \mathcal{E})$ as $f^*(\mathcal{I}, \mathcal{E})$:*

$$f^*(\mathcal{I}, \mathcal{E}) = \max_{\mathcal{J} \in V} \underline{f}(\mathcal{I} \cup \mathcal{J}) \tag{16}$$

where V is the set of feasible combination of unfixed points that may be included:

$$V = \left\{ \mathcal{J} \subseteq \mathcal{U} | \nexists k \in \mathcal{E} : \mathbf{x}^k \in B(\mathcal{I} \cup \mathcal{J}) \right\}. \tag{17}$$

A first approximation of the upper bound is given by the next lemma.

Lemma 3. *Consider a node $n(\mathcal{I}, \mathcal{E})$ and that the inference process of included points is perfect: $\mathcal{I} = \{i \in \{1, ..., M\} | \mathbf{x}^i \in B(\mathcal{I})\}$. Then,*

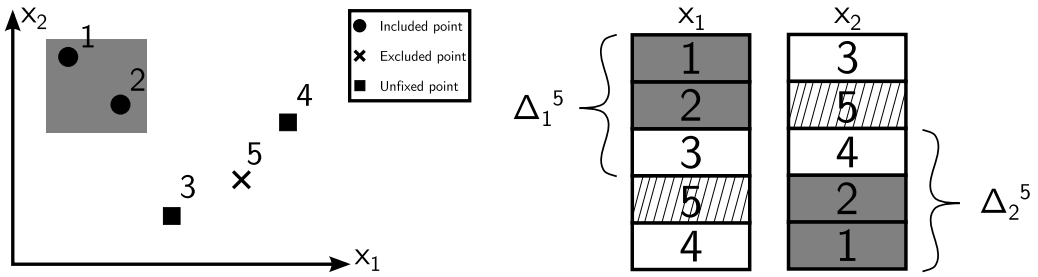
$$f^*(\mathcal{I}, \mathcal{E}) \leq \underline{f}(\mathcal{I}) + \sum_{i \in \mathcal{U} \cap \mathcal{P}} c^i. \tag{18}$$

Proof. For every node of the subtree of root $n(\mathcal{I}, \mathcal{E})$, the maximum score that can be achieved is given by summing up the values of all the unfixed positive points:

$$f^*(\mathcal{I}, \mathcal{E}) \leq \sum_{i \in \mathcal{I}} c^i + \sum_{i \in \mathcal{U} \cap \mathcal{P}} c^i = \underline{f}(\mathcal{I}) + \sum_{i \in \mathcal{U} \cap \mathcal{P}} c^i. \tag{19}$$

□

In the example given by Figure 5a, we have five points where three are fixed. Let us assume that they are all positive except point 5 which is excluded. As there are two unfixed points, the simple upper bound of Lemma 3 would be the sum of the point 1, 2, 3 and 4.



(a) Exemple with 5 points where the point 5 is excluded.

(b) The points are sorted along the two dimensions x_1 and x_2. Two set of points excluding the point 5 can be defined.

Figure 5: Example of a node where the points are sorted along each dimension.

A tighter upper bound can be inferred. We first explain how it works in the example of Figure 5. First we sort the points by their coordinates for each dimension as illustrated in Figure 5b. If we consider the excluded point 5, it is obvious that we can further include 3 or 4 but not both of them. There must be at least one dimension t where the excluded

point is not included in $B_t(\mathcal{I})$. In the example, this means that the upper bound is given by the sum of 1, 2 and 3 in the first dimension, and by the sum of 4, 2 and 1 in the second dimension. To take into account both cases, we consider the maximum over these two sums.

We now define a quantity which corresponds to the sum of the positive points that can be included if the point i is excluded in the dimension t.

Definition 4. *Consider a node $n(\mathcal{I}, \mathcal{E})$. For each excluded point $i \in \mathcal{E}$ and dimension $t \in \{1, ..., D\}$, we define the value Δ_t^i which depends of the fixing.*

- *If $\mathcal{I} \neq \emptyset$,*

  1. *If $l_t > x_t^i$ then*
  $$\Delta_t^i = \sum_{j \in \mathcal{U} \cap \mathcal{P} : x_t^j > x_t^i} c^j \tag{20}$$

  2. *If $u_t < x_t^i$ then*
  $$\Delta_t^i = \sum_{j \in \mathcal{U} \cap \mathcal{P} : x_t^j < x_t^i} c^j \tag{21}$$

  3. *If $x_t^i \in [l_t, u_t]$ then*
  $$\Delta_t^i = 0 \tag{22}$$

- *If $\mathcal{I} = \emptyset$,*
$$\Delta_t^i = \max \left\{ \sum_{j \in \mathcal{U} \cap \mathcal{P} : x_t^j < x_t^i} c^j, \sum_{j \in \mathcal{U} \cap \mathcal{P} : x_t^j > x_t^i} c^j \right\}. \tag{23}$$

Lemma 4. *Consider a node $n(\mathcal{I}, \mathcal{E})$. For each excluded point $i \in \mathcal{E}$ and dimension t, Δ_t^i is the maximum sum of the positive points that can further be included if the point $i \in \mathcal{E}$ is excluded in the dimension $t \in \{1, ..., D\}$.*

Proof. If $\mathcal{I} \neq \emptyset$, an operational box exists. In the dimension t, the operational box defines a range $[l_t, u_t]$. If the coordinate x_t^i is in the range $[l_t, u_t]$, the point i can not be excluded in the dimension t because it is already included by the operational box in this dimension. In this case, we choose to set $\Delta_t^i = 0$.

On the other hand, if the coordinate x_t^i is on one side of the range $[l_t, u_t]$: $x_t^i < l_t$ or $x_t^i > u_t$, every point on the same side of the range can be included. In these cases, the value of Δ_t^i is given respectively by (20) and (21).

If $\mathcal{I} = \emptyset$ no operational box is defined. The point i could be excluded in the dimension t either by including only points with coordinates less than x_t^i or by including only points with coordinates greater than x_t^i. The maximum sum of the positive points in this case must consider the two possibilities as in (23). □

The values Δ_t^i provides a tighter upper bound which is given by Lemma 5. The complete procedure to obtain a tighter upper bound is summarized in Algorithm 3.

Lemma 5. *Consider a node $n(\mathcal{I}, \mathcal{E})$ and that the inference process of included points is perfect: $\mathcal{I} = \{i \in \{1, ..., M\} | \mathbf{x}^i \in B(\mathcal{I})\}$. Then,*

$$f^*(\mathcal{I}, \mathcal{E}) \leq \underline{f}(\mathcal{I}) + \min_{i \in \mathcal{E}} \left\{ \max_{t \in \{1, ..., D\}} \Delta_t^i \right\}. \tag{24}$$

Proof. Consider an excluded point $i \in \mathcal{E}$, there must be at least one dimension t such that the point i is excluded in this dimension. In particular, the point i may be excluded in a way that includes a maximum of positive points. By Lemma 4, the maximum sum of points that can further be included if the point i is excluded is given by $\max_{t\in\{1,...,D\}} \Delta_t^i$.

As every excluded point $i \in \mathcal{E}$ defines a maximum sum of points that can be included if i is excluded, an upper bound is given by the sum of the current score and the smallest maximum sum of points. □

Algorithm 3 Upper bound computation

$\Delta \leftarrow \sum_{i\in\mathcal{U}\cap\mathcal{P}} c^i$
for all $j \in \mathcal{E}$ and j is not inferred to be excluded **do**
  $\Delta^j \leftarrow 0$
  for all $t = 1, ...D$ **do**
    if $\mathcal{I} = \emptyset$ **or** $l_t > x_t^j$ **then**
      $\Delta^j \leftarrow \max\{\Delta^j, \sum_{i\in\mathcal{U}\cap\mathcal{P}:x_t^i>x_t^j} c^i\}$
      if $\Delta^j \geq \Delta$ **then break**
    end if
    if $\mathcal{I} = \emptyset$ **or** $u_t < x_t^j$ **then**
      $\Delta^j \leftarrow \max\{\Delta^j, = \sum_{i\in\mathcal{U}\cap\mathcal{P}:x_t^i<x_t^j} c^i\}$
      if $\Delta^j \geq \Delta$ **then break**
    end if
  end for
  $\Delta \leftarrow \min\{\Delta, \Delta^j\}$
end for
$\overline{f} \leftarrow \underline{f} + \Delta$

Δ_t^i can be computed quickly by sorting each point in each dimension by their coordinates as a pre-calculation. As a result, the algorithmic complexity of computing the upper bound at each node in the search tree is proportional to $D|\mathcal{E}|$. To reduce the computational cost of computing the upper bound, the following lemma can be exploited, which is already included in Algorithm 3.

Lemma 6. *Consider a node $n(\mathcal{I}, \mathcal{E})$ and $\mathcal{I} \neq \emptyset$. Points inferred to be excluded can be ignored to compute the upper bound of Lemma 5.*

Proof. Consider a point i inferred to be excluded by an excluded point $j \in \mathcal{E}$: $j \in B(\mathcal{I}\cup\{i\})$. Note that this inference implies that $\mathcal{I} \neq \emptyset$. In the upper bound given by the Lemma 5, the inferred excluded point i can be ignored if

$$\underline{f}(\mathcal{I}) + \max_{t\in\{1,...,D\}} \Delta_t^j \leq \underline{f}(\mathcal{I}) + \max_{t\in\{1,...,D\}} \Delta_t^i. \tag{25}$$

We now prove that $\forall t \in \{1, ..., D\}$,

$$\Delta_t^i \geq \Delta_t^j. \tag{26}$$

Following Definition 4, there are three cases to consider as $\mathcal{I} \neq \emptyset$.

1. If $l_t > x_t^j$, $\Delta_t^j = \sum_{k\in\mathcal{U}\cap\mathcal{P}:x_t^k>x_t^j} c^k$. Point i is inferred to be excluded by j and $l_t > x_t^j$ implies that $x_t^i < x_t^j \leq l_t$ by Lemma 1. Therefore,

$$\Delta_t^i = \sum_{k\in\mathcal{U}\cap\mathcal{P}:x_t^k>x_t^i} c^k \geq \Delta_t^j \tag{27}$$

2. Similarly if $u_t < x_t^j$, $\Delta_t^i \geq \Delta_t^j$.

3. If $x_t^j \in [l_t, u_t]$, $\Delta_t^j = 0$. If $x_t^i \in [l_t, u_t]$, $\Delta_t^i = 0 = \Delta_t^j$. If $x_t^i \notin [l_t, u_t]$, $\Delta_t^i \geq 0$ and $\Delta_t^i \geq \Delta_t^j$.

In all three cases, $\Delta_t^i \geq \Delta_t^j$ and (25) holds. □

3.3. Branching strategy

We now consider the branching strategy which determines which point to branch on at every iteration of Algorithm 1. Two factors need to be taken into account: the number of nodes resulting from a strategy and the computation burden to take a branching decision. In this paper, we investigate three strategies, two classical coming from standard techniques for linear programming based branch-and-bound and one designed for our problem.

1. Full strong branching

2. Reliability branching

3. Least local branching.

Strong branching [3, 15] is a well known mixed integer programming technique. The idea is to test a subset of all possible branching and to select the one leading to the best decrease of the upper bound. Full strong branching [2] refers to the case were we investigate the whole set of possibility of branching. This strategy works very well in practice with respect to the number of nodes computed but not enough to compensate the high computation time per node. Applying this method to our formulation is straightforward and is summarized in Algorithm 4.

Algorithm 4 Strong branching for a node $n(\mathcal{I}, \mathcal{E})$

$\forall j \in \mathcal{U}$, compute $s_j = \texttt{score}\left(\sum_{k\in\mathcal{P}} c^k - \overline{f}(\mathcal{I} \cup \{j\}, \mathcal{E}), \sum_{k\in\mathcal{P}} c^k - \overline{f}(\mathcal{I}, \mathcal{E} \cup \{j\})\right)$.
Branch on the variable $j \in \mathcal{U}$ with the highest score s_j

Branching candidates are compared thanks to a single value. This value is a score which captures the results of including or excluding a point.

Definition 5. *The predicate score used in this paper, suggested by [1], is defined as:*

$$\texttt{score}(a, b) = (a + \epsilon)(b + \epsilon). \tag{28}$$

We adapt the reliability branching introduced by Achterberg, Koch and Martin in [2]. This branching rule is a combination between strong branching and pseudocost branching [4]. Pseudocost branching keeps a history of branching decisions. The cost of branching on a variable i is calculated as $\texttt{score}(P_i^+, P_i^-)$ where P_i^+ and P_i^- are the per-unit changes in the objective function value [15]. In mixed integer programming, the fractional part of the variable is used to compute P_i^+ and P_i^-. In our case, the upper bound is not computed through a linear relaxation. Therefore, we need to define a version of P_i^+ and P_i^- for the algorithm presented in this paper.

Definition 6. *Consider a point* $i \in \{1, ..., M\}$*, we define the* per-unit changes in the objective function value *as*

$$P_i^+ = \sum_{k \in \mathcal{P}} c^k - \frac{F_i^+}{\eta_i} \text{ and } P_i^- = \sum_{k \in \mathcal{P}} c^k - \frac{F_i^-}{\eta_i} \tag{29}$$

where η_i *is the number of problems where the algorithm branched on* i *and* F_i^+ *(*F_i^-*) is the sum of the* η_i *upper bounds of these problems where* i *is set to be included(excluded).*

Reliability branching branches on the variable i with the highest pseudocost if the size of the historic η_i is large enough. If the historic is too small, the cost of branching on this variable is not reliable enough and strong branching is used to update the score of the variable. We use $\eta_{rel} = 10$ as a minimal historic size to be reliable. To limit the number of strong branching, a look ahead strategy [2] can be used. If no new best candidate point was found for λ successive strong branching, the best candidate is kept as a final choice. In our implementation, we use $\lambda = 16$.

Algorithm 5 Reliability branching for a node $n(\mathcal{I}, \mathcal{E})$

$\forall j \in \mathcal{U}$, compute $s_j = \texttt{score}(F_j^+, F_j^-)$.
Sort **s** in decreasing order
$n \leftarrow 0$
$maxS \leftarrow 0$
for all $j \in \mathcal{U} : \eta_j < \eta_{rel}$ **and** $n < \lambda$ **do**
  Try branching on j
  Update η_j, F_j^+, F_j^- and s_j
  if $s_j > maxS$ **then**
    $maxS \leftarrow s_j$, $n \leftarrow -1$
  end if
  $n \leftarrow n + 1$
end for
Branch on the variable $j \in \mathcal{U}$ with the highest score s_j

Finally, we introduce our least local branching strategy. We compute an approximation of the upper bound. In the case of the inclusion of a point i, the strategy only computes the score of the new box. In the case of the exclusion of i, we only compute the values $\Delta_t^i \ \forall t \in \{1, ..., D\}$ defined in Section 3.2 to obtain an approximation of the new upper bound. Based on this information, the same technique as strong branching is applied: the strategy computes a score and selects the one with the highest score. The branching can be slightly improved if we consider the following observation:

Observation 4. *Consider a node* $n(\mathcal{I}, \mathcal{E})$*,* $\underline{f}_g$ *the global lower bound and a point* $i \in \mathcal{U}$*. If* $\overline{f}(\mathcal{I} \cup \{i\}, \mathcal{E}), \overline{f}(\mathcal{I}, \mathcal{E} \cup \{i\}) \leq \underline{f}_g$ *the branch can be pruned. If* $\overline{f}(\mathcal{I} \cup \{i\}, \mathcal{E}) \leq \underline{f}_g$ *and* $\overline{f}(\mathcal{I}, \mathcal{E} \cup \{i\}) > \underline{f}_g$*, we can branch on this point and exclude it.*

The complete procedure is summarized in Algorithm 6. The performance of the different branching strategies are demonstrated in Section 4.

Algorithm 6 Least local branching for a node $n(\mathcal{I}, \mathcal{E})$

candidate $\leftarrow \emptyset$, candidate score $\leftarrow -1$
for all $i \in \mathcal{U}$ **do**
  Let Δ_t^j the minimum value computed for the upper bound in Algorithm 3 for one excluded point $j \in \mathcal{E}$ in dimension t.
  if $c^i > 0$ **and** $\big((x_t^j < l_t$ **and** $x_t^j < x_t^i)$ **or** $(u_t > x_t^j$ **and** $x_t^i < x_t^j)\big)$ **then**
    $\Delta_t^j \leftarrow \Delta_t^j - c^i$
  end if
  $s^+ \leftarrow \underline{f}(\mathcal{I} \cup \{i\}) + \Delta_t^j$
  $s^- \leftarrow \underline{f}(\mathcal{I}) + \min\{\Delta_t^j, \max_{t\in\{1,\dots,D\}}\{\Delta_t^i\}\}$.
  if $s^+ \leq \underline{f}_g$ **and** $s^- \leq \underline{f}_g$ **then**
    $\rightarrow$ Prune by bound
  else if $s^+ \leq \underline{f}_g$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$
  end if
  $s \leftarrow \texttt{score}(\sum_{k\in\mathcal{P}} c^k - s^+, \sum_{k\in\mathcal{P}} c^k - s^-)$
  if $s >$ candidate score **then**
    candidate $\leftarrow i$, candidate score $\leftarrow s$
  end if
end for

4. Computational Results

The algorithm is tested on 300 databases. The points are uniformly generated with coordinates between 0.0 and 1.0. Their values c^i are uniformly assigned between 1.0 and -1.0. We compare our combinatorial algorithm with *CPLEX 12.6* with default parameters (including presolve, cutting planes, ...) and limited to one core only. We refer to the first MILP formulation of Section 2.1 by *C-MILP* and to the pure binary one of Section 2.2 by *B-MILP*. We observed that using *C-MILP* or *B-MILP* on a single core lead to lower running time than allowing it to run on the eight available cores of the computer. Our algorithm is also implemented to run on a single core. We consider branching either on the list of positive points or on the whole set of points. We refer to the first case by the abbreviation *BoPP* and to the second case by *BoP*. To denote the different branching strategies discussed in Section 3.3, we use *St* for strong branching, *R* for reliability branching and *LL* for least local branching. Concerning the node to pop from the workpile at each iteration, our implementation chooses the node with the highest primal solution $\underline{f}$. It can be empirically observed that this method quickly finds a good lower bound which results in the lowest computation time.

Table 1 reports the final gap, the time taken and the number of nodes processed for the tests. These results have been obtained on an *Intel Core i7* with a clock rate of 3.47 GHz and 24 GB of RAM. Each line gives the geometric mean of 10 tests. Each of these tests has limit running time of one hour. The final gap is computed with respect to the upper bound $\overline{f}$ provided by each algorithm:

$$\text{gap} = \frac{\overline{f} - \underline{f}}{\overline{f}}. \tag{30}$$

At the end of the table are reported the total number of problems solved, the relative geometric mean time and the relative geometric mean number of nodes processed.

D	M	BoPP - St	BoPP - R	BoPP - LL	BoP - R	BoP - LL	C-MILP	B-MILP
		Gap [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]
		Time [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]
		Nodes	*Nodes*	*Nodes*	*Nodes*	*Nodes*	*Nodes*	*Nodes*
10	100	–	–	–	–	–	–	–
		5.93	**1.13**	1.48	2.18	4.48	21.00	8.65
		12728	49811	23019	41302	11442	5847	608
10	125	–	–	–	–	–	–	–
		79.04	22.52	**17.87**	31.66	58.74	183.63	20.91
		104088	653296	188315	407871	113933	43147	1583
10	150	–	–	–	–	–	–	–
		379.15	97.45	53.98	109.93	242.12	710.48	**51.73**
		298859	2472799	419301	1129545	330304	141960	3763
10	175	0.12	–	–	–	0.05	1.30	–
		1976.49	677.89	344.44	1094.68	1785.22	3266.15	**198.39**
		1340387	12352777	2548846	8305828	2073103	629511	13855
10	200	20.55	0.06	0.06	0.30	9.14	16.55	–
		3354.09	1855.21	1198.39	2646.13	2977.74	3605.57	**176.39**
		2054754	34419637	6797263	20214041	2901915	477170	9523
12	100	–	–	–	–	–	–	–
		10.16	**2.47**	2.59	3.21	8.60	34.73	10.37
		22990	109076	37449	59464	19401	8451	905
12	125	–	–	–	–	–	–	–
		84.78	**21.70**	25.38	31.11	68.47	163.61	41.20
		119299	736356	280719	453429	110884	27217	2825
12	150	0.02	–	–	0.02	0.02	0.07	–
		574.75	188.61	129.48	300.37	615.20	1121.90	**94.91**
		544395	5212430	1112644	3291999	719104	146968	5828
12	175	1.61	0.68	–	0.69	0.71	1.80	–
		2833.94	1540.73	1028.95	2042.74	2936.84	2734.06	**296.74**
		1821636	31246739	6286113	16700693	2910439	329483	19193
12	200	22.04	21.45	1.68	21.59	22.27	22.75	–
		3561.28	3291.81	2545.31	3260.21	3547.70	3604.35	**761.72**
		1810123	57928659	14448348	22870067	2845269	423166	37094
14	100	–	–	–	–	–	–	–
		29.99	**4.05**	4.97	7.14	19.53	54.13	17.92
		50719	178603	72655	128712	33230	9342	1517
14	125	–	–	–	–	–	–	–
		117.31	39.45	**23.11**	39.87	101.04	272.22	56.19
		149336	1247202	242070	551635	128381	29856	3800
14	150	0.60	0.05	–	0.05	0.12	0.31	–
		1399.21	612.48	409.34	762.66	1267.84	1452.77	**151.23**
		1201436	15100459	3028122	7956344	1442966	145623	9290
14	175	3.35	0.66	0.29	1.57	0.68	1.49	–
		3060.28	1975.16	1346.73	2543.84	3218.04	3139.37	**386.65**
		1865464	38630100	7792205	20596769	2823951	314093	17647
14	200	39.10	7.73	3.42	17.87	42.46	3.54	–
		3600.02	3237.75	3274.80	3406.47	3600.01	3289.68	**477.40**
		1687939	56737957	15358677	21995182	2794774	228352	15849
16	100	–	–	–	–	–	–	–
		5.76	1.71	2.98	**1.34**	6.76	16.85	12.23
		13432	74957	36897	23301	10507	3341	729
16	125	–	–	–	–	–	–	–
		97.85	**23.02**	29.10	33.97	102.61	174.73	61.48
		136704	768937	317596	484505	118167	21009	3581
16	150	0.12	0.02	–	0.02	0.12	0.15	–
		1440.51	904.25	544.76	624.73	1510.89	1304.95	**247.73**
		1100314	16728707	3713333	6723082	1490820	99414	10053

D	M	BoPP - St	BoPP - R	BoPP - LL	BoP - R	BoP - LL	C-MILP	B-MILP
		Gap [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]	*Gap* [%]
		Time [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]	*Time* [*s*]
		Nodes	*Nodes*	*Nodes*	*Nodes*	*Nodes*	*Nodes*	*Nodes*
16	175	15.04	3.22	1.46	1.45	17.61	2.70	–
		3385.32	2882.24	2298.37	2661.95	3420.72	3322.80	**503.22**
		2244577	59320468	13190269	21831692	2837793	208126	21820
16	200	46.52	44.92	20.50	47.17	46.53	21.95	**0.04**
		3600.02	3600.02	3502.60	3600.01	3600.01	3603.99	1437.08
		1625834	63138068	17062590	22705446	2615809	225811	60286
18	100	–	–	–	–	–	–	–
		6.36	**1.58**	3.85	2.05	8.52	24.87	16.95
		14025	68800	47521	36501	14121	3186	1015
18	125	–	–	–	–	–	–	–
		174.87	42.88	64.32	**35.69**	104.35	207.00	67.95
		222788	1280361	483770	495022	115341	20197	3236
18	150	0.25	0.05	0.02	0.05	0.12	0.04	–
		2391.84	878.76	1045.15	1379.46	2042.90	1561.50	**312.26**
		1921759	21252578	7029154	13884012	1761012	113233	11869
18	175	14.69	3.24	17.27	7.70	17.42	5.96	–
		3211.44	2723.00	3186.22	2808.95	3289.46	3132.31	**774.62**
		2049049	54091394	16042225	21928761	2574547	140577	26493
18	200	40.38	38.58	42.25	41.66	41.62	17.02	**0.04**
		3600.02	3600.02	3600.02	3600.01	3600.01	3603.89	1554.10
		1750244	63511756	15682977	22579255	2574981	152513	54359
20	100	–	–	–	–	–	–	–
		18.14	3.84	10.36	**3.68**	15.33	48.70	40.16
		41515	165125	117307	62440	19485	7569	2124
20	125	0.02	–	–	–	–	–	–
		931.63	310.91	341.76	290.03	651.67	687.12	**184.01**
		1026276	8886256	2774937	3489913	680707	49355	9469
20	150	0.24	–	–	–	0.26	0.03	–
		1529.21	720.49	652.79	824.56	1702.51	1096.03	**287.83**
		1227191	17438179	4091615	8541028	1411501	68994	10664
20	175	35.17	37.39	39.05	39.50	37.79	9.48	**0.02**
		3600.01	3600.02	3600.02	3600.01	3600.01	3603.51	1258.91
		2374808	69571096	18419495	23894124	2753152	157941	50818
20	200	35.78	35.57	17.56	37.16	38.60	7.05	**0.02**
		3600.01	3600.01	3511.04	3600.01	3600.01	3556.24	1638.34
		1575877	62549797	14502356	23657342	2411663	113486	51131
Solved /300		**180**	**212**	**227**	**207**	**189**	**179**	**294**
Mean time		**3.21**	**1.36**	**1.32**	**1.60**	**2.93**	**4.50**	**1.00**
Mean nodes		**57.39**	**669.91**	**193.00**	**321.64**	**62.16**	**9.00**	**1.00**

Table 1: Results table in function of the number of dimensions D and the number of points M.

In the results of Table 1, the pure binary MILP formulation clearly outperforms the other methods. The B-MILP method solves 294 instances out of the 300 given, processes less nodes than the others and takes the least amount of time. The pure binary MILP formulation is taken as comparison reference for the others.

Second in line comes the branching on positive points with the least local branching strategy. This method solves 227 problems. The geometric mean time is 32% greater than pure binary MILP formulation even though the number of nodes processed is 193 times higher. Using the reliability branching strategy increases a little the computation

time and multiplies the number of nodes to process by three. Using the strong branching strategy decreases the number of nodes processed drastically. Still, the number of nodes processed is more than 50 times higher than the number processed by CPLEX with the B-MILP formulation.

Branching on the whole set of points leads to processing on average half the number of nodes but with an increased computation time. The increase can be explained by the complexity to compute the upper bound which is proportional to the number of excluded points. If the algorithm branches on the whole set of points, the number of excluded points is potentially bigger as well as the theoretical maximal depth of the branch-and-bound tree.

To conclude, the binary formulation is the best exact method to solve the box search problem. However, the algorithm that we propose in this paper is a simple and dependency-free alternative that may be implemented quickly.

Acknowledgments

This research is supported by the public service of Wallonia within the framework of the VIRTUOSO project. The authors thank the companies Segal and AGC for providing the industrial databases used in this paper.

References

[1] Achterberg, T., 2009. Scip: solving constraint integer programs. Mathematical Programming Computation 1, 1–41. doi:10.1007/s12532-008-0001-1.

[2] Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Operations Research Letters 33, 42–54. doi:10.1016/j.orl.2004.04.002.

[3] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J., 2006. The Traveling Salesman Problem. Princeton University Press, Princeton.

[4] Bénichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribière, G., Vincent, O., 1971. Experiments in mixed-integer linear programming. Mathematical Programming 1, 76–94.

[5] Chong, I.G., Jun, C.H., 2008. Flexible patient rule induction method for optimizing process variables in discrete type. Expert Systems with Applications 34, 3014–3020. doi:10.1016/j.eswa.2007.05.047.

[6] Dobkin, D.P., Gunopulos, D., Maass, W., 1996. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. Journal of Computer and System Sciences 52, 453–470. doi:10.1006/jcss.1996.0034.

[7] Durand, G., Szigeti, H., Nouedoui, L., Chaplais, C., 2010. Prim2: An extension of the prim algorithm to support smarter manufacturing for sustainable development. IMS2020-Proceedings from the IMS2020 Summer School on Sustainable Manufacturing , 249–272.

[8] Eckstein, J., Hammer, P., Liu, Y., Nediak, M., Simeone, B., 2002. The maximum box problem and its application to data analysis. Computational Optimization and Applications 23, 285–298. doi:10.1023/A:1020546910706.

[9] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., 1996. From data mining to knowledge discovery in databases. AI magazine 17, 37.

[10] Friedman, J.H., Fisher, N.I., 1999. Bump hunting in high-dimensional data. Statistics and Computing 9, 123–143.

[11] Goldberg, N., Shan, C.C., 2007. Boosting optimal logical patterns using noisy data, pp. 228–236.

[12] Knauer, C., 2010. The complexity of geometric problems in high dimension, in: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (Eds.), Theory and Applications of Models of Computation. Springer. volume 6108 of *Lecture Notes in Computer Science*, pp. 40–49. doi:10.1007/978-3-642-13562-0_5.

[13] Land, A.H., Doig, A.G., 1960. An automatic method of solving discrete programming problems. Econometrica 28, 497–520. doi:10.2307/1910129.

[14] Lavrač, N., Kavšek, B., Flach, P., Todorovski, L., 2004. Subgroup discovery with cn2-sd. The Journal of Machine Learning Research 5, 153–188.

[15] Linderoth, J., Savelsbergh, M., 1999. A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing 11, 173–187. doi:10.1287/ijoc.11.2.173.
[16] Liu, Y., Nediak, M., 2003. Planar case of the maximum box and related problems, in: Proceeding of the Canadian Conference on Computational Geometry, pp. 11–13.
[17] Pyle, D., 1999. Data preparation for data mining. volume 1. Morgan Kaufmann.
[18] Yukizane, T., Ohi, S.Y., Miyano, E., Hirose, H., 2006. The bump hunting method using the genetic algorithm with the extreme-value statistics. IEICE transactions on information and systems E89-D, 2332–2339. doi:10.1093/ietisy/e89-d.8.2332.