Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach

Guillaume Derval[a,*], Pierre Schaus[a]

[a] *UCLouvain, Belgium*

Abstract

A Maximal-Sum Submatrix (MSS) maximizes the sum of the entries corresponding to the Cartesian product of a subset of rows and columns from an original matrix (with positive and negative entries). Despite being NP-hard, this recently introduced problem was already proven to be useful for practical data-mining applications. It was used for identifying bi-clusters in gene expression data or to extract a submatrix that is then visualized in a circular plot. The state-of-the-art results for MSS are obtained using an advanced Constraint Programing approach that combines a custom filtering algorithm with a Large Neighborhood Search. We improve the state-of-the-art approach by introducing new upper bounds based on linear and mixed-integer programming formulations, along with dedicated pruning algorithms. We experiment on both synthetic and real-life data, and show that our approach outperforms the previous methods.

Keywords: Combinatorial optimization, Maximum-sum submatrix, Linear relaxation, Constraint programming

1. Introduction

The strengths of the relationships between two sets of objects can be encoded as a matrix. Such examples are the traffic between two sets of nodes in a computer network (Medina et al., 2002), the gene expressions for a set of patients (Van't Veer et al., 2002), the bilateral migration between two sets of countries (The World Bank, 2018; Dao et al., 2018), ranked tiling (Le Van et al., 2014), etc.

When the set of objects is large, mining such matrix manually to understand the structure of the relations is not an easy task. One important question is that of summarizing the most important relationships. The Maximal-Sum Submatrix (MSS) problem has been introduced in (Branders et al., 2017) to answer this question. A MSS maximizes the sum of the entries corresponding

*Corresponding author

Email addresses: guillaume.derval@uclouvain.be (Guillaume Derval), pierre.schaus@uclouvain.be (Pierre Schaus)

to the Cartesian product of a subset of rows and columns from an original matrix (with positive and negative entries). The size of the MSS can be controlled by a priori subtracting a common constant from all the entries. In this setting the MSS can be viewed as a (more or less compact) summary of the most important relations between two subsets of rows and columns. As pointed in Branders et al. (2017), the MSS problem shares similarities with the biclustering one (Hartigan, 1972; Madeira & Oliveira, 2004) attempting to discover homogeneous submatrices rather than heavy ones. Biclustring techniques have been mainly applied to bioinformatics.

Solving the MSS problem exactly is an NP-Hard problem (Branders et al., 2017), as it encodes the maximum edge-weighted biclique problem (Peeters, 2003). It is actually a bipartite form of the Quadratic Pseudo-Boolean Optimization problem (QPBO) (Rother et al., 2007). This bipartite form is also known as BQPBO (Punnen et al., 2015).

The state-of-the-art results for MSS are obtained using an advanced Constraint Programming (CP) approach that combines a custom filtering algorithm with a Large Neighborhood Search (LNS) (Shaw, 1998). We improve and extend the state-of-the-art approach by introducing two new bounding and search tree pruning strategies. First, we show how a linear program (LP) relaxation of a mixed integer linear formulation can be solved in linear time with a dedicated algorithm, without using complex algorithms such as the simplex, but rather by inspection. We then use this linear-programming relaxation as an upper-bounding procedure to cut off the search tree and derive the exact reduced costs to prune the values of the variables in a global constraint following the reduced-cost based filtering idea of Focacci et al. (1999). Second, we demonstrate the use of Lagrangian relaxation for this problem, by finding another set of tighter upper bound, although more costly to compute.

We then experiment on both synthetic and real-life data showing the significant speedups obtained with the new hybrid LP-CP approach.

2. Definitions and notations

This paper uses multiset notations. Braces $\{\}$ are used for sets and brackets $[]$ for multisets. Sets and multisets are both represented as uppercase characters (S). Vector and matrices are represented as bold characters, respectively lowercase and uppercase ($\boldsymbol{x}$, $\boldsymbol{M}$). Elements of vectors and matrices, and scalars in general, are represented as normal italic characters (x_i, M_{ij}, i).

Let $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ be a matrix with both positive and negative real numbers. The set of rows and columns of the matrix are defined as $L_R := \{1, \ldots, m\}$, $L_C := \{1, \ldots, n\}$, respectively.

If $I \subseteq L_R$ and $J \subseteq L_C$ are subsets of the rows and of the columns, respectively, $\boldsymbol{M}_{I,J}$ denotes the submatrix of $\boldsymbol{M}$ that contains only the elements M_{ij} belonging to the submatrix with set of rows I and set of columns J. Throughout this paper, i is always an index of a row, and j is always an index of a column.

Definition 1. ***The Maximal-Sum Submatrix Problem.*** *The Maximal-Sum Submatrix (MSS) is the submatrix* $\boldsymbol{M}_{R^*,C^*}$*, with* $R^* \subseteq L_R$ *and* $C^* \subseteq L_C$*, such that:*

$$(R^*, C^*) = \underset{I \subseteq L_R, J \subseteq L_C}{\operatorname{argmax}} \sum_{i \in I, j \in J} M_{i,j} \tag{1}$$

Example 1. *Given the following matrix:*

$$\boldsymbol{M}^{ex} = \begin{array}{c} \begin{array}{ccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \left[\begin{array}{ccccccc} -3 & -1 & 3 & -1 & 2 & -3 & 1 \\ -2 & -2 & 3 & -3 & 3 & 0 & -2 \\ 0 & 2 & 0 & 1 & -2 & 2 & 0 \\ 0 & 0 & 2 & -3 & 2 & -2 & 1 \\ -3 & 2 & -3 & 0 & 0 & 2 & -2 \\ -1 & 1 & -1 & 2 & 1 & 1 & -3 \\ -2 & 1 & 0 & 2 & -2 & 2 & -2 \\ 1 & -2 & -2 & 1 & -1 & -2 & -3 \end{array}\right] \end{array} \begin{array}{c} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \end{array}$$

The maximal-sum submatrix of $\boldsymbol{M}^{ex}$ *is* $\boldsymbol{M}^{ex}_{\{3,5,6,7\},\{2,4,6\}}$ *(highlighted in black), its value being 18.*

An important property of the MSS identified in (Branders et al., 2017) is that the search space can be limited to the selection of the subset of columns or to the subset of rows as stated in the next observation:

Observation 1. *Given a fixed subset of columns* $C \subseteq L_C$*, an optimal subset of rows for the MSS is the one computed by*

$$R^* = \{i \in L_R \mid \sum_{j \in C} M_{ij} > 0\} \tag{2}$$

3. Existing work, similar problems and variants

The Maximum-Sum submatrix problem is related to other problems, some of them being equivalent. This section lists existing work made on similar problems.

3.1. Maximum Weighted Edge Biclique (MWEB)

The input matrix of the MSS can be seen as an adjacency matrix of a bipartite weighted graph (which is actually a biclique). In this context, finding the MSS is equivalent of finding a biclique whose sum of the edges' weight is maximum.

Definition 2. ***Maximum Weighted Edge Biclique (MWEB) problem*** *(Tan, 2008) Given a complete bipartite graph* $G = (V_1, V_2, E)$*, and an edge weighting function* $w_G : E \to \mathbb{R}$*, find a biclique* $(A \subseteq V_1, B \subseteq V_2)$ *such that the sum of the weight of the edges in the biclique is maximal.*

Tan (2008) discusses this problem and shows that it is inapproximable (for a problem of size $m \times n$, no polynomial time algorithm can approximate MWEB within a factor $\max(m,n)^\epsilon$ for $\epsilon > 0$, unless RP=NP). This result thus also holds for MSS. The unweighted version of the problem, the Maximum Edge Biclique (MEB), is NP-Hard (Peeters, 2003). It has been used to describe many variants of the biclustering problem (see Tanay et al. (2002) for an example, or the survey from Madeira & Oliveira (2004)).

3.2. Multiple maximum submatrices

One may want to extract multiple submatrices from a given dataset. Multiple methods have been introduced to this end, that differ in how they handle overlaps between found submatrices.

Branders et al. (2019a) introduced the Maximum Weighted Set of Disjoint Submatrices Problem (MWSDSP), that amounts at finding K disjoint submatrices such that the sum of their content is maximized:

$$\underset{(I^1,I^2,\ldots,I^k),(J^1,J^2,\ldots,J^k)}{\operatorname{argmax}} \sum_{k=1}^{K} \sum_{i \in I^k, j \in J^k} M_{ij} \tag{3}$$

$$\text{subject to } (I^k \times J^k) \cap (I^{k'} \times J^{k'}) = \emptyset \ \forall k, k' \in [1,K], k \neq k' \tag{4}$$

The solving method used in that paper is a column generation (Desaulniers et al., 2006) one, using an MSS solver as the column generator (columns are, in the context, actually candidate submatrices) and a MIP to select the optimal submatrices among the candidate ones.

The disjointedness constraint may be too strong in some context (in patient/gene matrices, a gene can be involved in multiple illnesses, and a patient can have multiple illnesses too, for example). Another variation which allows submatrices to overlap but do not count them multiple times is known as the Maximum Weighted Submatrix Coverage Problem (MWSCP) (Derval et al., 2019):

$$\underset{(I^1,I^2,\ldots,I^k),(J^1,J^2,\ldots,J^k)}{\operatorname{argmax}} \sum_{i,j \mid \exists k:(i,j) \in I^k \times J^k} M_{ij} \tag{5}$$

that is, the problems amounts at finding K submatrices such that the sum of the content of their union is maximal.

In another work, Branders et al. (2019b) use a greedy variant of MWSCP to find biclusters in gene expression data. They use an MSS solver multiple time, each time removing the newly found submatrix from the main matrix and replacing it with zeros. They show that the biclusters they found using this technique (called K-CPGC) are biologically relevant.

3.3. Other biclustering algorithms

Biclustering is a broad subject, and numerous methods have been created. Most of them attempt to find homogeneous biclusters in some sense. A survey by Madeira & Oliveira (2004), focusing on biological data, separates the methods in four families:

- Biclusters with constant values
- Biclusters with constant values on rows or columns (each row/column can have a different, but fixed, value)
- Biclusters with coherent values (additive or multiplicative models, ...)
- Biclusters with coherent evolution (values are evolving inside a row/column following a given model)

Note that the MSS does not fall into any of these categories. They further subdivide these methods depending on the number of bicluster they find (single, multiple and disjoint/disjoint per row/disjoint per column/non-overlapping/non-overlapping with tree structure), and classify 19 different problems and methods inside these categories. Forty-nine methods are reviewed and classified in (Pontes et al., 2015). Another survey by Xie et al. (2018) focuses on the applicability of the biclustering algorithms in biological and biomedical data.

Some pattern mining algorithms can also be viewed as biclustering binary matrices. Frequent Itemset Mining aims at finding frequent itemsets inside a dataset of transaction. Each transaction contains a list of items. The dataset can then be represented as a binary matrix, with transactions as rows, items as columns, and a 1 in cells where the transaction contains a given item. Frequent itemsets are itemsets that are present as a subset of a given number (given a priori) of transactions. The task, in biclustering terms, then amounts at finding large submatrices filled with ones.

Most methods in the literature focus on finding all closed (that cannot be extended) frequent itemsets (see, for example, (Agrawal et al., 1996; Han et al., 2004) for dedicated methods, or (Schaus et al., 2017) for a Constraint Programming-based method).

Tiling (Geerts et al., 2004) aims at finding large *tiles* inside binary matrices, i.e. finding large submatrices containing only ones, as previously. Tiling differs from Frequent Itemset Mining as it focuses on the area of the tiles, not on frequency (in this context, frequency is equivalent to number of rows, while the area is the number of rows in the submatrix times the number of columns). Le Van et al. (2014) introduce an extension to matrices representing *ranks* (each row is a permutation of $1 \dots n$, giving a *rank* to each column) and aims at finding biclusters with similar ranks.

4. An upper bound solvable by inspection

A natural upper bound for this problem without limits on the numbers of selected rows / columns is the sum of the positive elements in $\boldsymbol{M}$:

$$\sum_{i,j} \max(M_{ij}, 0). \tag{6}$$

It was used by Branders et al. (2017) as the cut-off upper bound for their branch-and-bound algorithm. We present below an upper bounding procedure based

on a Big-M formulation of the problem(Griva et al., 2009). We then prove that this bound is tighter than the upper bound presented by Branders et al. (2017).

Branders et al. (2017) introduced a MIP model for MSS relying on observation 1 and using Big-M constraints. The main decision variables are the selection status of the rows/columns with binary variables r_i, c_j. The contribution of row i to the objective, p_i, is then

$$p_i = \begin{cases} \sum_j M_{ij} \cdot c_j & \text{if } r_i = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

This constraint is linearized with the big-M constants up_i (resp. lo_i) being the positive sum of the positive (resp. negative) contributions of the row i. The complete model (named $\mathrm{F_{BigM}}$ hereafter) is given next.

$$\max \textstyle\sum_i p_i \tag{8a}$$

$$p_i \leq r_i \cdot up_i \quad \forall i \tag{8b}$$

$$p_i \leq (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i \quad \forall i \tag{8c}$$

$$r_i, c_j \in \{0, 1\} \quad \forall i, j \tag{8d}$$

with $up_i = \sum_{j \in L_C} \max(M_{ij}, 0)\ \forall i$, and $lo_i = -\sum_{j \in L_C} \min(M_{ij}, 0)\ \forall i$ being respectively the upper bound and the opposite of the lower bound reachable contribution to the objective, for a given row i. Note that these bounds can be computed more precisely when we have partial states for the r_i and c_j variables (typically, while inside a search tree). We explore later this possibility (we call this variant of the model the *recompute* variant). It is however difficult to do in a MIP solver, as locally modifying the matrix coefficient is complex and costly.

Intuitively, if r_i is 0, i.e. the row i is not selected, then p_i must be 0; this is encoded by constraint (8b). The right part of constraint (8c) becomes $(\sum_j M_{ij} \cdot c_j) + lo_i$ which is greater than 0 by construction, and thus less constraining than (8b). If $r_i = 1$, then an upper bound for the contribution p_i is up_i (thus, in this case, (8b) is not constraining p_i). Constraint (8c) becomes $p_i \leq \sum_j M_{ij} \cdot c_j$ which is the contribution of the row i when selected.

Linear Programming Relaxation. By relaxing the integrality constraint on the rows of the MIP model $\mathrm{F_{BigM}}$ (8), i.e. using $r_i \in [0, 1]\ \forall i$, we obtain an LP model whose optimum provides an upper bound for the MSS. We call this particular version of the model the *row-relaxed* model, $\mathrm{F_{BigM\text{-}linear\text{-}rows}}$. It has interesting properties:

Theorem 4.1. *For any row i taken in isolation, and with column variables already selected (i.e. variables c_j fixed $\forall j$), the value r_i maximizing p_i in $F_{BigM\text{-}linear\text{-}rows}$ is*

$$r_i^* := \frac{lo_i + \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{9}$$

Proof. From (8b) and (8c), and as we must maximize p_i

$$p_i = \min(r_i \cdot up_i, (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i). \tag{10}$$

In this context, r_i is a continuous variable, only constrained by this particular relation. Variable p_i in function of r_i is a convex function (the minimum of two linear functions is a convex function), and its maximum is reached when both components are equal:

$$r_i \cdot up_i = (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i \tag{11}$$

$$r_i = \frac{lo_i + \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{12}$$

□

Moreover, the optimal contribution is thus

$$p_i^* = up_i \cdot r_i^* = \frac{up_i \cdot lo_i + up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{13}$$

These two properties can now be used to derive upper bounds for the MSS problem. This is our first contribution:

Theorem 4.2. *$F_{BigM\text{-}linear\text{-}rows}$ has an optimal objective of*

$$\sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_j \max\left(0, \sum_i \frac{up_i \cdot M_{ij}}{up_i + lo_i}\right). \tag{14}$$

This value thus provides an upper-bound for F_{BigM} which is equivalent to the MSS problem.

Proof. The only constraints in the MIP formulation where the variable r_i appears are (8b) and (8c), meaning that rows are effectively independent from each other: with a given set of selected columns, the optimal value for p_a will not change if r_b changes $\forall a \neq b \in L_R$. We have thus that for each row i, $r_i = r_i^*$ (see Theorem 4.1). The objective becomes

$$\sum_i p_i = \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_i \frac{up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i} \tag{15}$$

$$= \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_j c_j \cdot (\sum_i \frac{up_i \cdot M_{ij}}{up_i + lo_i}) \tag{16}$$

By inspection, this expression is maximized with $c_j = 1$ (resp. 0) if $\sum_i \frac{up_i \cdot M_{ij}}{up_i + lo_i} > 0$ (resp. < 0). □

As stated in the next theorem, this bound is tighter than the sum of the positive contributions.

Theorem 4.3. *The bound obtained from the optimal solution of $F_{BigM\text{-}linear\text{-}rows}$ is tighter than the simple sum of the positive contributions $\sum_i up_i$.*

Proof. From (15).

$$\sum_i p_i = \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_i \frac{up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i} \leq \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_i \frac{up_i^2}{up_i + lo_i} = \sum_i up_i$$

□

It is also non-symmetric, the minimum between the upper bound of the matrix and of its transpose can thus be taken.

Example 2. *Given the matrix $\boldsymbol{M} = \left(\begin{smallmatrix} 3 & 0 \\ -6 & 6 \end{smallmatrix}\right)$, the sum of the positive contributions is 9, the upper bound obtained from $F_{BigM\text{-}linear\text{-}rows}$ is 6, while it is 7 with $\boldsymbol{M}^T$.*

5. A Lagrangian-based upper bounding procedure

The model $\mathrm{F_{BigM\text{-}linear\text{-}rows}}$ is not the only way to model the MSS problem as a MIP. A more straightforward model $\mathrm{F_x}$ which uses more variables (notably one variable per cell) is presented below:

$$\max \sum_{i \in L_R, j \in L_C} M_{ij} \cdot x_{ij} \tag{17a}$$

$$x_{ij} \leq r_i \quad \forall i, j \tag{17b}$$

$$x_{ij} \leq c_j \quad \forall i, j \tag{17c}$$

$$r_i + c_j \leq x_{ij} + 1 \quad \forall i, j \tag{17d}$$

$$r_i, c_j, x_{ij} \in \{0, 1\} \quad \forall i, j \tag{17e}$$

Rows and column selection are represented by variables r_i and c_j. x_{ij} indicates if the cell i, j is selected. Constraints (17b) and (17c) ensure that if a cell is selected, then the associated row and column are selected. Constraint (17d) ensures that if both a row and a column are selected, then the cell is selected.

Some constraints in this model are redundant. As it is a maximization problem, we have two cases:

- Either $M_{ij} > 0$ and the value x_{ij} will be maximized, and thus either constraint (17b) or (17c) will be tightened for this particular (i, j);
- Or $M_{ij} < 0$ and the value x_{ij} will be minimized, thus tightening constraint (17d).

Constraints (17b), (17c) and (17d) can thus be rephrased without loss of generality as

$$x_{ij} \leq r_i \quad \forall i, j : M_{ij} > 0 \tag{18a}$$

$$x_{ij} \leq c_j \quad \forall i, j : M_{ij} > 0 \tag{18b}$$

$$r_i + c_j \leq x_{ij} + 1 \quad \forall i, j : M_{ij} < 0. \tag{18c}$$

The linear relaxation of $\mathrm{F_x}$ (named hereafter $\mathrm{F_{x\text{-}linear}}$) uses more variables than $\mathrm{F_{BigM\text{-}linear\text{-}rows}}$ ($\mathcal{O}(|L_R| \cdot |L_C|)$ rather than $\mathcal{O}(|L_R| + |L_C|)$) making this model more complex to use in an off-the-shelf MIP solver in practice, as running the simplex or other standard LP-solving algorithm is too slow or uses too much memory to be run at each node of the search tree on big matrices.

It is however possible to use a Lagrangian relaxation of $\mathrm{F_x}$ to obtain good upper bound of the optimal linear solution. We introduce Lagrange multipliers α_{ij}, β_{ij} and γ_{ij} respectively for constraints (18a), (18b) and (18c). For simplicity we add each of the three multipliers for each variable, but we set them to zero for non-existing constraints (i.e. $\alpha_{ij} = \beta_{ij} = 0$ if $M_{ij} \leq 0$ and $\gamma_{ij} = 0$ if $M_{ij} \geq 0$). The Lagrangian relaxation leads to the following model, $\mathrm{F_{x\text{-}lrelax\text{-}all}}$:

$$\min_{\alpha_{ij},\beta_{ij},\gamma_{ij}} \max_{r_i,c_j,x_{ij}} \sum_{ij} M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) + \beta_{ij}(c_j - x_{ij}) + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) \tag{19a}$$

$$\text{with} \quad r_i, c_j, x_{ij} \in \{0,1\} \quad \forall i,j \tag{19b}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_{ij} \geq 0 \quad \forall i,j \tag{19c}$$

$$\alpha_{ij} = \beta_{ij} = 0 \quad \forall i,j : M_{ij} \leq 0 \tag{19d}$$

$$\gamma_{ij} = 0 \quad \forall i,j : M_{ij} \geq 0 \tag{19e}$$

The constraint (19b) can be changed to a linear version ($\in [0,1]$) without any modification to the optimal solutions, and is thus equivalent to its linear relaxation. By the Strong Lagrangian Duality property (Boyd et al., 2004) (that implies that Lagragian relaxations of convex problems respecting Slater conditions, which holds in the *linear* problem $\mathrm{F_{x\text{-}linear}}$, have the same objective value as the original problem), its optimum is thus the same as $\mathrm{F_{x\text{-}linear}}$.

The maximization part is a convex function on the Lagrangian multipliers $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$: we optimize their values using a sub-gradient algorithm. However the high number of parameters leads to a slow convergence in practice.

We show below another Lagrangian relaxation, $\mathrm{F_{x\text{-}lrelax\text{-}partial}}$, which does not relax all the constraints and as consequence uses less multipliers. Usually this prevents a simple, inspection-like solving of the Lagrangian subproblem but in this particular case we demonstrate it can be solved easily.

$$\min_{\alpha_{ij},\beta_{ij},\gamma_{ij}} \max_{r_i,c_j,x_{ij}} \sum_{ij} M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) \tag{20a}$$

$$\text{with} \quad x_{ij} \leq c_j \quad \forall i,j \tag{20b}$$

$$r_i, c_j, x_{ij} \in \{0,1\} \quad \forall i,j \tag{20c}$$

$$\alpha_{ij}, \gamma_{ij} \geq 0 \quad \forall i,j \tag{20d}$$

$$\alpha_{ij} = 0 \quad \forall i,j : M_{ij} \leq 0 \tag{20e}$$

$$\gamma_{ij} = 0 \quad \forall i,j : M_{ij} \geq 0 \tag{20f}$$

Again it can be shown that the constraint (20c) can be relaxed into a linear version while not modifying the optimum solutions, and by the same argument

as before, is then equivalent to $\mathrm{F}_{\text{x-linear}}$. The same subgradient method can be used as the Lagrangian subproblem is still convex on the Lagrangian multipliers. However, for a given set of Lagrangian multiplier, the Lagrangian subproblem cannot be solved as trivially as before. Let us give a name to the Lagrangian subproblem, and rearrange its formulation:

$$f(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \max_{r_i, c_j, x_{ij}} \sum_{i,j} M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) \tag{21}$$

$$= \max_{r_i, c_j, x_{ij}} \sum_{i,j} \big(x_{ij} \cdot (M_{ij} - \alpha_{ij} + \gamma_{ij}) + \gamma_{ij}\big) + \sum_i r_i \cdot (\sum_j \alpha_{ij} - \gamma_{ij}) - \sum_j c_j \cdot (\sum_i \gamma_{ij}) \tag{22}$$

$$\text{such that } x_{ij} \leq c_j \ \forall i, j$$

Selecting the optimal rows r_i for a given set of multiplier is trivial ($r_i = 1$ if $\sum_j \alpha_{ij} - \gamma_{ij} > 0$, $r_i = 0$ otherwise). The case for x_{ij} and c_j is more complex. Two cases are possible for each specific column j:

- Either $c_j = 0$. In that case, by constraint (23) all cells on this column are unselected: $x_{ij} = 0 \ \forall i$. The overall contribution of these variables is thus 0.
- Or $c_j = 1$. In that case, all cells on this column are selected depending on whether their contributions are positive or not: $x_{ij} = 1 \iff M_{ij} - \alpha_{ij} + \gamma_{ij} > 0$. The overall contribution of these variables is then

$$\sum_i \max(0, M_{ij} - \alpha_{ij} + \gamma_{ij}) - \gamma_{ij}. \tag{23}$$

We can thus conclude that in any optimal solution, a column will be selected ($c_j = 1$) if $\sum_i \max(0, M_{ij} - \alpha_{ij} + \gamma_{ij}) - \gamma_{ij} > 0$, and will not otherwise. This provides a linear time algorithm (in the size of the matrix) to compute the Lagrangian subproblem, by directly applying the resulting formula:

$$f(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \sum_i \max(0, \sum_j \alpha_{ij} - \gamma_{ij}) + \sum_j \max(0, \sum_i \max(0, M_{ij} - \alpha_{ij} + \gamma_{ij}) - \gamma_{ij}) + \sum_{i,j} \gamma_{ij} \tag{24}$$

Overall, the subproblem of $\mathrm{F}_{\text{x-lrelax-partial}}$ can be solved in the same complexity as $\mathrm{F}_{\text{x-lrelax-all}}$ (i.e. $\mathcal{O}(|L_R| \cdot |L_C|)$) but with fewer Lagrangian multipliers, ensuring a faster convergence for the subgradient algorithm, while preserving the exact same optimal objective value. The algorithm to minimize $\mathrm{F}_{\text{x-lrelax-partial}}$ is given in Algorithm 1.

Algorithm 1 Solving $F_{\text{x-lrelax-partial}}$

```
function solveSubproblem(α, γ)
    r_i ← 0 ∀i ∈ L_R                          ▷ indicates if row i is selected or not
    c_j ← 0 ∀j ∈ L_C                          ▷ indicates if column j is selected or not
    x_ij ← 0 ∀i ∈ L_R, j ∈ L_C                ▷ indicates if cell i, j is selected or not
    ub ← 0                                    ▷ The upper bound being computed

    for all i ∈ L_R, j ∈ L_C do               ▷ Base cell contribution
        ub ← ub + γ_ij

    for all i ∈ L_R do                        ▷ Select all rows with positive contribution
        contribution ← Σ_j α_ij − γ_ij
        if contribution > 0 then
            r_i ← 1
            ub ← ub + contribution

    for all j ∈ L_C do                        ▷ Select all columns with positive contribution
        contribution ← 0
        for all i ∈ L_R do                    ▷ Contribution of a column includes the ones of its cells
            contribution ← contribution − γ_ij
            cellContribution ← M_ij − α_ij + γ_ij
            if cellContribution > 0 then
                contribution ← contribution + cellContribution
                x_ij ← 1

        if contribution > 0 then
            ub ← ub + contribution
            c_j ← 1
        else                                  ▷ If, at the end of the computation, we do not select the
            for all i ∈ L_R do                  column, we must reset the cells to 0.
                x_ij ← 0

    return ub, r, c, x

function gradientDescent(nIterations)
    α, γ ← random initialization (uniform between 0 and 1)
    μ ← 1
    for nIterations iterations do
        ub, r, c, x ← solveSubproblem(α, γ)   ▷ Note that here, at any point of the algorithm,
        μ ← 0.95 · μ                            ub is a valid upper bound
        for all i ∈ L_R do
            for all j ∈ L_C do
                α_ij ← max(0, α_ij − μ · (r_i − x_ij))
                γ_ij ← max(0, γ_ij − μ · (x_ij + 1 − r_i − c_j))
    return solveSubproblem(α, γ)
```

The algorithm uses a simple subgradient-descent algorithm with a harmonic update (Cambazard & Fages, 2015) of step size μ. By default we use a value of 0.95 for the update, and start with $\mu_0 = 1$. The optimality gap between the optimum of $F_{\text{x-linear}}$ and the one computed by the subgradient algorithm on random matrices filled with random Gaussian noise is shown in Figure 1. We experimentally observe on that Figure that we obtain convergence in about 150 iterations even on large matrices, and thus we limit the number of iterations to this number.

Experiments are also run with 50 and 100 iterations, and with a mechanism that only update the multipliers each q nodes (details are in the experiment section). Existing research shows that even poor approximations or non-updated approximation of the Lagrangian multipliers may be beneficial (Sellmann, 2004).

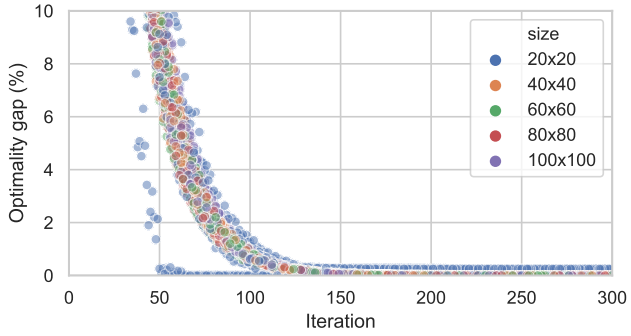

Figure 1: Optimality gap between $\mathrm{F}_{\text{x-linear}}$ and the value computed by the subgradient on 50 random matrices.

The computed bounds will be used to cut in the branch-and-bound tree. If by misfortune the upper bound found is not accurate (i.e. is greater than expected), the tree will simply visit more nodes, but will not give an invalid result, as the bounds are valid.

6. A note about bounds' strengths and relations

So far we discussed three different bounding procedure:

- The natural upper bound $\sum_{i,j} \max(M_{ij}, 0)$;
- The $\mathrm{F}_{\text{BigM-linear-rows}}$ model, presented in section 4, based on a Big-M relaxation that uses one variable per row and column. We show above that it is solvable by inspection in linear time;
- The $\mathrm{F}_{\text{x-linear}}$ model presented in section 5, which uses one variable per cell, row, and column. We show above how to use Lagrange multipliers to solve it.

These bounds are in fact in increasing order of strength. We showed in theorem 4.3 that $\mathrm{F}_{\text{BigM-linear-rows}}$ is a tighter bound than the natural one. The following theorem shows that $\mathrm{F}_{\text{x-linear}}$ is tighter than $\mathrm{F}_{\text{BigM-linear-rows}}$.

Theorem 6.1. *The optimal objective of* $\mathrm{F}_{\text{x-linear}}$ *is less or equal than the optimal objective of* $\mathrm{F}_{\text{BigM-linear-rows}}$.

Proof. Given an optimal solution $(r_i, c_j \forall i, j)$ of $\mathrm{F}_{\text{x-linear}}$, we show that using the same values for r_i and c_j for all rows and columns gives a greater solution in $\mathrm{F}_{\text{BigM-linear-rows}}$, which implies its optimum is greater.

The optimal variables x_{ij} for $\mathrm{F}_{\text{x-linear}}$ can be inferred from the row/column variables. From equations (17b), (17c) and (17d) we have the following constraints on x_{ij}:

$$\max(0, r_i + c_j - 1) \leq x_{ij} \leq \min(r_i, c_j) \tag{25}$$

As the objective maximizes $\sum_{ij} M_{ij} \cdot x_{ij}$, we have the following:

- $x_{ij} = \min(r_i, c_j)$ if $M_{ij} > 0$
- $x_{ij} = \max(0, r_i + c_j - 1)$ if $M_{ij} < 0$

The optimal objective for $\mathrm{F}_{\text{x-linear}}$ is then

$$\mathrm{obj}_{\text{x-linear}} = \sum_i \Big(\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \Big) \tag{26}$$

We can insert this solution (r_i, c_j) inside $\mathrm{F}_{\text{BigM-linear}}$ (note that $\mathrm{F}_{\text{BigM-linear}}$ and $\mathrm{F}_{\text{BigM-linear-rows}}$ have the same solutions).

From equations (8a), (8b) and (8c), we have that the solution is

$$\mathrm{obj}_{\text{BigM-linear}} = \sum_i \min(r_i \cdot up_i, (\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i) \tag{27}$$

If we take individually each row from $\mathrm{obj}_{\text{x-linear}}$:

$$\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \tag{28}$$

$$\leq \sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} \tag{29}$$

$$\leq \sum_{j|M_{ij}>0} r_i \cdot M_{ij} = r_i \cdot up_i \tag{30}$$

Moreover,

$$\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \tag{31}$$

$$\leq \sum_{j|M_{ij}>0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i + c_j - 1) \cdot M_{ij} \tag{32}$$

$$\leq \sum_{j|M_{ij}>0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i - 1) \cdot M_{ij} \tag{33}$$

$$\leq \sum_j c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i - 1) \cdot M_{ij} = \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i \tag{34}$$

All of this for any row i. We thus have that for each row, the contribution of the row in $\mathrm{obj}_{\text{x-linear}}$ is less or equal than in $\mathrm{obj}_{\text{BigM-linear}}$. From equations (30)

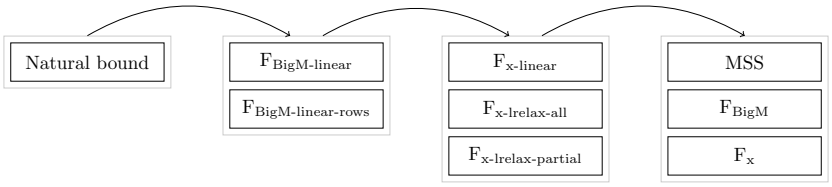

Figure 2: Summary of all the models used in this paper. A gray rectangle means that enclosed methods are equivalent (have the same optimum objective). An arrow from A to B indicates that B has a lower optimum than A. This relation is transitive.

and (34):

$$\begin{aligned}
&\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \\
&\leq \min(r_i \cdot up_i, \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i) \ \forall i \qquad (35) \\
\implies &\sum_i \sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \\
&\leq \sum_i \min(r_i \cdot up_i, \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i) \qquad (36) \\
\implies &\text{obj}_{\text{x-linear}} \leq \text{obj}_{\text{BigM-linear}} \qquad (37)
\end{aligned}$$

□

It is thus expected that bounding and filtering based on $\text{F}_{\text{x-linear}}$ will prune the search space more than ones based on $\text{F}_{\text{BigM-linear}}$. Figure 2 gives a visual representation of all the bounds used in this paper.

It is possible to show that all these linear bounds can be arbitrarily distant from the discrete problem's optimal solution.

Example 3. *Let $\boldsymbol{M}^{ex2}$ be an $n \times n$ matrix with positive values in the diagonal but negative ones everywhere else:*

$$\boldsymbol{M}^{ex2} = \begin{pmatrix} a & -b & -b & \dots & -b \\ -b & a & -b & \dots & -b \\ -b & -b & a & & -b \\ \vdots & \vdots & & \ddots & \vdots \\ -b & -b & -b & \dots & a \end{pmatrix}$$

with $a, b \in \mathbb{R}^+$. The optimal MSS objective value is a by construction. Table 1 shows the bounds obtained by all linear bounding methods presented above.

As an admissible solution is to select all the cells in the best row/column and leave the rest unselected, we have that the bounds found by these linear

Table 1: Bounds obtained for example 3

Method	Value for (n, a, b) $(20, 1, 1000)$	$(20, 19, 1)$
Natural bound	20	380
$F_{\text{BigM-linear}}$	$\simeq 19.9989$	190
$F_{\text{x-linear}}$	10	190
Optimum MSS	1	19

relaxations can be at most $\min(m, n)$ times greater than the optimum discrete objective in an $m \times n$ matrix.

Moreover, the bound found by all the linear relaxations above will be at least $\frac{\sum_{i,j} \max(M_{ij},0)}{2}$ as the solution $r_i = c_j = \frac{1}{2}$ is always admissible in $F_{\text{x-linear}}$, and produce the aforementioned objective value. This can be at most $\frac{\min(m,n)}{2}$ times greater than the actual discrete optimum.

7. Using the bounds in a CP framework

We reuse the framework introduced by (Branders et al., 2017). It is based on Constraint Programming which, in this particular case, summarizes to a Depth First Search using Branch-and-Bound on the space of the possible assignment of the variables (rows and columns to select).

We use one binary variable per row $(r_1, \ldots, r_m)$ and per column $(c_1, \ldots, c_n)$. These binary variables can either be assigned ($v = 1$), unassigned ($v = 0$) or undecided yet ($v = \perp$). At each DFS iteration, an undecided variable is selected (according to a *search strategy*, described in the following subsections), and the DFS branches on the left by assigning this variable to 1, and on the right to 0. The various bounds are then computed, and the filtering rules are applied, that is the removal of impossible values from the domains of the variables. It is, of course, possible that a domain becomes empty, making the solver backtrack in the search tree.

Let us define $R^s = \{i \in L_R \mid r_i = s\}$ and $C^s = \{j \in L_C \mid c_j = s\}$, the sets of variable currently having a given status $s \in \{\perp, 0, 1\}$. Note that the sets $R^0, R^1, R^\perp$ are disjoint and that their union equals L_R, the same being true for columns. We thus omit most of the time one of these three sets, mainly R^0, as its value can be computed intuitively as $L_R \setminus (R^1 \cup R^\perp)$. We also denote from this point R and C as any choice of rows and columns (respectively) that can be an improving solution.

7.1. Dealing with partial solutions

Without loss of generality, all partial assignments of variables for any matrix $\boldsymbol{M}$ can be reduced to a partial solution with $|R^1| = |C^1| = 1$ and $|R^0| = |C^0| = 0$ on a transformed matrix $\boldsymbol{M}_s$. Let $\gamma_1, \ldots, \gamma_{|R^\perp|}$ be any ordering of rows $\in R^\perp$,

and θ an ordering of the columns $\in C^\perp$. We can construct the matrix $\boldsymbol{M_s} \in \mathbb{R}^{(|R^\perp|+1)\times(|C^\perp|+1)}$ such that:

$$M_{s,ij} = \begin{cases} \sum_{a\in R^1}\sum_{b\in C^1} M_{ab} & \text{if } i=j=1 \\ \sum_{a\in R^1} M_{a\theta_{j+1}} & \text{if } i=1 \text{ and } j>1 \\ \sum_{b\in C^1} M_{\gamma_{i+1}b} & \text{if } j=1 \text{ and } i>1 \\ M_{\gamma_{i+1}\theta_{j+1}} & \text{otherwise} \end{cases}$$

$$\begin{array}{c} \begin{array}{ccccc} C^1 & \theta_1 & \theta_2 & \cdots & \theta_{|C^\perp|} \end{array} \\ \left(\begin{array}{ccccc} & \multicolumn{4}{c}{\sum \text{ of sel. rows}} \\ \sum & \multicolumn{4}{c}{\text{Remaining}} \\ \text{of} & \multicolumn{4}{c}{\text{cells}} \\ \text{sel.} & \multicolumn{4}{c}{\text{of old}} \\ \text{cols} & \multicolumn{4}{c}{\text{matrix}} \end{array}\right) \end{array} \begin{array}{c} R^1 \\ \gamma_1 \\ \gamma_2 \\ \cdots \\ \gamma_{|R^\perp|} \end{array}$$

i.e. we merge selected rows into a single one by summing them, remove the excluded ones from the matrix, and keep all the other ones as they are, and do the same with the columns.

One of our contributions is the usage of this simplification trick during the search. As we use a Large Neighborhood Search Strategy (Shaw, 1998), most of the variables are decided during the search.

7.2. Update of the incumbent solution

As observed in (Branders et al., 2017), it is not necessary to wait until all the decision variables are decided in order to update the incumbent (best so far) solution and the lower-bound. By construction, each node of the search tree can be transformed greedily in linear time into an admissible partial solution to possibly become the incumbent solution of the branch and bound.

$C = C^1$ is a partial solution with $R \subseteq R^1 \cup R^\perp$ such that the objective is maximum. Such R can be computed easily using Observation 1. Precisely, the objective value for this solution is

$$\sum_{i\in R^1}\sum_{j\in C^1} M_{ij} + \sum_{i\in R^\perp} \max(0, \sum_{j\in C^1} M_{ij}) \tag{38}$$

The selected columns are thus only composed of the ones already selected, discarding the undecided ones to build the optimal solution. The transpose reasoning is also true[1].

The algorithm used to maintain the incumbent solution is below. It maintains for each column the sum of the already-selected entries $\sum_{i\in R^1} M_{ij}$ (and similarly for each row the sum of the selected entries), and use these caches to compute the possible new solution objective value faster.

The whole update needed to find a possible new incumbent solution is in $\mathcal{O}(n+m)$.

7.3. Upper bounding

The upper bounds introduced earlier must be adapted to take into account partial assignments of decision variables.

[1]Note that (Branders et al., 2017) did not consider this transpose reasoning (for the unconstrained cardinality case) in the update rule for the incumbent, and that the two solutions are in general different.

Algorithm 2 Maintain incumbent solution

$\texttt{rowVal}[i] = \sum_{j \in C^1} M_{ij} \ \forall i \in L_C \setminus C^0$
$\texttt{colVal}[j] = \sum_{i \in R^1} M_{ij} \ \forall j \in L_R \setminus R^0$

function onRowModified(i) ▷ Called when a row is selected/unselected ($\perp \rightarrow 0/1$)
  if row becomes selected **then**
    for all $j \in L_C \setminus C^0$ **do** ▷ No need to do it for non-selectable columns
      $\texttt{colVal}[j] \leftarrow \texttt{colVal}[j] + M_{ij}$
  updateIncumbentSolution()

function onColumnModified(j) ▷ Called when a column is selected/unselected ($\perp \rightarrow 0/1$)
  if column becomes selected **then**
    for all $i \in L_R \setminus R^0$ **do** ▷ No need to do it for non-selectable rows
      $\texttt{rowVal}[i] \leftarrow \texttt{rowVal}[i] + M_{ij}$
  updateIncumbentSolution()

function updateIncumbentSolution()
  $\texttt{solR} \leftarrow 0$
  $\texttt{solC} \leftarrow 0$
  for all $i \in R^1$ **do**
    $\texttt{solR} \leftarrow \texttt{solR} + \texttt{rowVal}[i]$
  for all $j \in C^1$ **do**
    $\texttt{solC} \leftarrow \texttt{solC} + \texttt{colVal}[j]$
  for all $i \in R^\perp$ **do**
    $\texttt{solR} \leftarrow \texttt{solR} + \max(0, \texttt{rowVal}[i])$
  for all $j \in C^\perp$ **do**
    $\texttt{solC} \leftarrow \texttt{solC} + \max(0, \texttt{colVal}[j])$
  $\texttt{incumbentSolution} \leftarrow \max(\texttt{incumbentSolution}, \texttt{solR}, \texttt{solC})$

7.3.1. $F_{BigM\text{-}linear\text{-}rows}$

Equation (14), providing the optimal solution for $\mathrm{F}_{\text{BigM-linear-rows}}$ and thus a valid upper bound for the MSS problem, can be adapted to the following:

$$\sum_{i \in R^\perp} \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_{j \in C^1} \left(\sum_{i \in R^1} M_{ij} + \sum_{i \in R^\perp} \frac{up_i \cdot M_{ij}}{up_i + lo_i} \right) + \sum_{j \in C^\perp} \max\left(0, \sum_{i \in R^1} M_{ij} + \sum_{i \in R^\perp} \frac{up_i \cdot M_{ij}}{up_i + lo_i}\right) \quad (39)$$

up_i and lo_i can also be adapted, as they need to be respectively the upper and minus the lower bound of the contribution of a row. Then, this definition is also valid:

$$up_i := \sum_{j \in C^1} M_{ij} + \sum_{j \in C^\perp} \max(0, M_{ij}) \quad (40)$$

$$lo_i := -\sum_{j \in C^1} M_{ij} + \sum_{j \in C^\perp} \max(0, -M_{ij}) \quad (41)$$

Recomputing up_i, lo_i for all rows along with the upper bound at each node of the search tree has a runtime of $\mathcal{O}(mn)$ per node (which, over a full branch, amounts to $\mathcal{O}(mn^2)$). This version of the upper-bounding procedure is shown below in the experiments as "$\mathrm{F}_{\text{BigM-linear-rows}}$ (recompute)".

A computational speed-up can be obtained by using caching and fixing up_i and lo_i before starting the computation (typically to $\sum_{j \in L_C} \max(0, M_{ij})$ and $\sum_{j \in L_C} \max(0, -M_{ij})$). While this reduces the pruning power of the bound (which will be less tight as a consequence), the increase in visited nodes (due to

a faster computation) can be worth the trade-off. In this case, the computation can be made in $\mathcal{O}(\Delta_r \cdot n + \Delta_c)$ at each node, where Δ_r is the number of modified row variables in the current tree node (and similarly for Δ_c). Over a full branch of the tree search, this sums up to $\mathcal{O}(mn)$. The algorithm is shown in Algorithm 3. The name of this upper bounding procedure is "$\mathrm{F}_{\text{BigM-linear-rows}}$ (fixed)" in the experiments presented in the last section.

Algorithm 3 Implementation of $\mathrm{F}_{\text{BigM-linear-rows}}$ (fixed) with caching

▷ Initialization
`curBound` $\leftarrow 0$ ▷ Reversible (i.e. reverted to previous value when a backtrack occurs)
`curColBound`$[j] \leftarrow 0\ \forall j$ ▷ Reversible

for all $i \in L_R$ **do**
  $up_i \leftarrow \sum_{j \in L_C} \max(0, M_{ij})$
  $lo_i \leftarrow \sum_{j \in L_C} \max(0, -M_{ij})$
  `rowContribution`$[i] \leftarrow \frac{up_i \cdot lo_i}{up_i + lo_i}$
  for all $j \in L_C$ **do**
    `cellContribution`$[i, j] \leftarrow \frac{M_{ij} \cdot lo_i}{up_i + lo_i}$
    `curColBound`$[j]$ $\leftarrow$ `curColBound`$[j]$ + `cellContribution`$[i, j]$
  `curBound` $\leftarrow$ `curBound` + `rowContribution`$[i]$

for all $j \in L_C$ **do**
  if `curColBound`$[j] > 0$ **then**
    `curBound` $\leftarrow$ `curBound` + `curColBound`$[j]$

function onColModified(i) ▷ Called when a column is selected/unselected ($\perp \rightarrow 0/1$)
  if $j \in C^1 \wedge$ `curColBound`$[j] < 0$ **then**
    ▷ If the column is now selected, but was not as its contribution is negative, add it.
    `curBound` $\leftarrow$ `curBound` + `curColBound`$[j]$
  else if $j \in C^0 \wedge$ `curColBound`$[j] > 0$ **then**
    ▷ If the column is now unselected, but had a positive contribution, remove it.
    `curBound` $\leftarrow$ `curBound` − `curColBound`$[j]$

function onRowModified(i) ▷ Called when a row is selected/unselected($\perp \rightarrow 0/1$)
  `curBound` $\leftarrow$ `curBound` − `rowContribution`$[i]$
  `delta` $\leftarrow 0$
  for all $j \in C^1 \cup C^\perp$ **do**
    `oldValue` $\leftarrow$ `curColBound`$[j]$
    `cdelta` $\leftarrow$ `cellContribution`$[i, j]$ ▷ the delta between the current
    if $i \in R^1$ **then** ▷ and new contribution of the cell
      `cdelta` $\leftarrow$ `cdelta` + M_{ij}
    `curColBound`$[j]$ $\leftarrow$ `curColBound`$[j]$ + `cdelta`

    if $j \in C^1$ **then**
      ▷ If the column is selected, then its contribution is updated.
      `delta` $\leftarrow$ `delta` + `cdelta`
    else if `oldValue` $<= 0 \wedge$ `curColBound`$[j] > 0$ **then**
      ▷ If the column was not selected, but should be now, add its contribution.
      `delta` $\leftarrow$ `delta` + `curColBound`$[j]$
    else if `oldValue` $> 0 \wedge$ `curColBound`$[j] <= 0$ **then**
      ▷ If the column was selected, but is not anymore, remove its old contribution.
      `delta` $\leftarrow$ `delta` − `oldValue`
    else if `oldValue` $> 0 \wedge$ `curColBound`$[j] > 0$ **then**
      ▷ If the column was selected, and still is, update the contribution.
      `delta` $\leftarrow$ `delta` + `cdelta`
  `curBound` $\leftarrow$ `curBound` + `delta`

7.3.2. $F_{x\text{-}lrelax\text{-}partial}$

The model presented at equation (20) can be reused as-is, simply by adding additional constraints to include/exclude rows/columns ($r_i = 1\ \forall i \in R^1$, $r_i = 0\ \forall i \in R^0$, ...). This requires a small adaptation to subproblem solving in Algorithm 1, shown in Algorithm 4. The algorithm runs in $\mathcal{O}(kmn)$ at each node of the tree, where k is the number of subgradient descent steps.

Algorithm 4 Solving $F_{\text{x-lrelax-partial}}$, with partial solution support (differences underlined)

function SOLVESUBPROBLEM($\boldsymbol{\alpha}, \boldsymbol{\gamma}$)
  $r_i \leftarrow 0\ \forall i \in L_R$ ▷ indicates if row i is selected or not
  $c_j \leftarrow 0\ \forall j \in L_C$ ▷ indicates if column j is selected or not
  $x_{ij} \leftarrow 0\ \forall i \in L_R, j \in L_C$ ▷ indicates if cell i, j is selected or not
  ub $\leftarrow 0$ ▷ The upper bound being computed

  for all $i \in \underline{R^1 \cup R^\perp}$, $j \in \underline{C^1 \cup C^\perp}$ **do** ▷ Base cell contribution
    ub $\leftarrow$ ub $+ \gamma_{ij}$

  for all $i \in \underline{R^1 \cup R^\perp}$ **do** ▷ Select all rows with positive contribution
    contribution $\leftarrow \sum_j \alpha_{ij} - \gamma_{ij}$
    if contribution $> 0\ \underline{\vee i \in R^1}$ **then**
      $r_i \leftarrow 1$
      ub $\leftarrow$ ub + contribution

  for all $j \in \underline{C^1 \cup C^\perp}$ **do** ▷ Select all columns with positive contribution
    contribution $\leftarrow 0$
    for all $i \in \underline{R^1 \cup R^\perp}$ **do** ▷ Contribution of a column includes the ones of its cells
      contribution $\leftarrow$ contribution $- \gamma_{ij}$
      cellContribution $\leftarrow M_{ij} - \alpha_{ij} + \gamma_{ij}$
      if cellContribution $> 0\ \underline{\vee i \in R^1}$ **then**
        contribution $\leftarrow$ contribution + cellContribution
        $x_{ij} \leftarrow 1$

    if contribution $> 0\ \underline{\vee j \in C^1}$ **then**
      ub $\leftarrow$ ub + contribution
      $c_j \leftarrow 1$
    else ▷ If, at the end of the computation, we do not select the column, we must reset the cells to 0.
      for all $i \in L_R$ **do**
        $x_{ij} \leftarrow 0$

  return ub, $\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{x}$

7.4. Reduced-Cost-based filtering

7.4.1. Upper bound filtering

Given a current state $(R^1, R^\perp, C^1, C^\perp)$, we can compute for each row $i \in R^\perp$ its upper bound when selected or unselected (namely $ub^{R^1 \cup \{i\}, R^\perp \setminus \{i\}, C^1, C^\perp}$ and $ub^{R^1, R^\perp \setminus \{i\}, C^1, C^\perp}$).

We then have these filtering rules:

$$ub^{R^1 \cup \{i\}, R^\perp \setminus \{i\}, C^1, C^\perp} < \text{best} \Rightarrow i \notin R \tag{42}$$

$$ub^{R^1, R^\perp \setminus \{i\}, C^1, C^\perp} < \text{best} \Rightarrow i \in R \tag{43}$$

i.e. if the upper bound for a particular assignment of a row is worse than the current best solution found, then this assignment is not part of any better

solution. The same is done for columns. The method is called cost-based domain filtering (Focacci et al., 1999).

Reusing at each node the upper bounding algorithm presented in the previous sections would require a too consequent increase in complexity; however, it is possible to do this pruning efficiently at no additional (asymptotic) cost.

This involves maintaining during the computation of the main upper bound the delta that would occur if the row/column variable is assigned to a specific value. While this is straightforward in practice, we do not include the code in the text of this paper for the sake of conciseness. The reference implementation is however available.

In practice, all pruning algorithms run in $\mathcal{O}(nm)$ (which is equivalent to the running time of the associated-bounding procedure), but for the fixed version of $\mathrm{F}_{\text{BigM-linear-rows}}$ (see Section 7.3.1). For this particular constraint, the column filtering occurs in $\mathcal{O}(m)$ (which is the running time of the upper bounding procedure) while the row filtering is in $\mathcal{O}(nm)$ (so, greater than the upper bounding procedure running time). We thus propose two versions of this pruning while associated with $\mathrm{F}_{\text{BigM-linear-rows}}$: one with the row filtering, one without.

7.4.2. Lower bound filtering

up_i and lo_i are, for a given row i, the upper bound and the opposite of the lower bound of its contribution. A version of these bounds which supports partial solutions is presented in equations (40) and (41).

The following filtering rules always apply[2]:

$$up_i < 0 \Rightarrow i \notin R^* \tag{44}$$

$$lo_i < 0 \Rightarrow i \in R^* \tag{45}$$

where R^* is the set of rows of one of the optimum solutions reachable from the current partial solution.

That is, if the maximum contribution of a row is negative, it will never be part of any (locally) optimal solution (reachable from this partial solution). Similarly, if the minimum contribution of a row is positive, then it will always be part of the best solution reachable from the current partial solution.

The rule (44) was introduced by (Branders et al., 2017), while (45) is a new contribution. This is straightforward, but was not possible in the implementation of (Branders et al., 2017) as it was not able to force rows to be in solution. As their constraint is included in all our experiments, we implemented the lower-bound filtering inside their constraint, with the option of being deactivable.

7.5. Methods and complexities summary

Table 2 provides a summary of all the methods and their complexities. Table 3 shows all the combinations of methods used in the experiments.

[2]Recall that lo_i is **the opposite** of the lower bound, hence the $<$.

Table 2: Methods and complexities (computed at each node, non-symmetric)

Model	Type	Upper bounding	Filtering		
			LB-Low	Row UB	Column UB
Base (Branders et al., 2017)		$\mathcal{O}(n\Delta_r + \Delta_c)$	$\mathcal{O}(\Delta_r + \Delta_c)$	$\mathcal{O}(mn\Delta)$	N/A
$F_{\text{BigM-linear-rows}}$	fixed	$\mathcal{O}(n\Delta_r + \Delta_c)$	N/A	$\mathcal{O}(mn\Delta)$	$\mathcal{O}(n\Delta)$
	recompute	$\mathcal{O}(mn\Delta)$	N/A	$\mathcal{O}(m\Delta)$	$\mathcal{O}(n\Delta)$
$F_{\text{x-lrelax-partial}}$		$\mathcal{O}(kmn\Delta)$	N/A	$\mathcal{O}(n\Delta)$	$\mathcal{O}(m\Delta)$

Δ_r and Δ_c are the number of modified rows/columns in the node. $\Delta = \Delta_r + \Delta_c$. Matrices are of size $m \times n$ and the Lagragian based method makes k steps. The complexities of the filtering operations are given **after** the upper bounding has been done, as it fills memoization arrays, allowing faster filtering.

Table 3: Combinations of methods used in the experiments

Name	Overall complexity	Applied models	Enabled filtering		
			LB-Low	Row UB	Col UB
Natural	$\mathcal{O}(mn\Delta)$	Base	✗	✓	-
Natural+LB+fast	$\mathcal{O}((m+n)\Delta)$	Base	✓	✗	-
Natural+LB	$\mathcal{O}(mn\Delta)$	Base	✓	✓	-
Fixed-BigM	$\mathcal{O}((m+n)\Delta)$	Base	✓	✗	-
		$F_{\text{BigM-linear-rows}}$ (fixed)	-	✗	✓
Recompute-BigM	$\mathcal{O}(mn\Delta)$	Base	✓	✓	-
		$F_{\text{BigM-linear-rows}}$ (recompute)	-	✓	✓
Lagrangian (k, skip s%)	$\mathcal{O}(kmn\Delta)$	Base	✓	✓	-
		$F_{\text{x-lrelax-partial}}$	-	✓	✓

✓: activated, ✗: disabled, -: non-applicable. Note that the models are always also applied in a symmetrical way (on the transpose matrix), but the table shows the enabled features for the non-symmetric case.

These combinations are chosen to show the relative improvement of each of our contributions, compared to the existing techniques. The pair (*Natural*, *Natural+LB*) aims at comparing the addition of the additional lower bound procedure introduced in Section 7.4.2. *Natural+LB+fast* provides a counterpart to *Fixed-BigM* as they have the same overall complexity. The same goes for *Natural+LB* and *Recompute-BigM*. We test the Lagrangian-based method presented earlier with various parameters for the number of iterations (k) and the probably of skipping an update at each node of the search tree (s%). The chosen pairs (k, s) are $(50, 100\%)$, $(100, 100\%)$, $(150, 100\%)$, $(150, 30\%)$, $(150, 60\%)$, $(150, 90\%)$, $(150, 95\%)$.

8. Experiments

We first experiment on synthetic data, showing differences in efficiency between all the methods presented, and then experiment with real-life data. We compare against a MIP solver (Gurobi) when appropriate. The MIP solver is run with the BigM model from equation (8).

8.1. Complete search

We generated small instances (square matrices from size 10×10 to 30×30) that can be solved to optimality, and compare the number of nodes explored

by each method by computing the ratio between this number of nodes and the one visited by the *Natural* method (see table 3), our baseline, along with the runtime ratio. We generated two datasets, the first being square matrices filled with a Gaussian noise $\mathcal{N}(0.2, 1)$, the other with $\mathcal{N}(0, 1)$.

There is a phase transition when the expected value of any row/column is 0, as this is the tipping point between selection and non-selection of a row/column. Moreover, the expected value of any submatrix in a matrix filled with noise $\mathcal{N}(0, 1)$ is 0, and all solutions are similar and close to 0. This makes the $\mathcal{N}(0, 1)$ case particularly difficult. On the other hand, $\mathcal{N}(0.2, 1)$ is comparatively simpler as the expected size for the MSS is the whole matrix. This kind of matrix appears when using LNS to solve the instances, as when the search progresses, we refine the main matrix by removing some columns with a negative contribution, increasing the mean value in the row/column/matrix. This choice of dataset thus shows the two phases possible in the problem, both being relevant.

Table 4 shows the ratio for all datasets and matrix sizes. As can be seen, the various methods proposed all improve over the state-of-the-art, but in the single case where only the new lower-bound filtering is activated. Fixed-BigM improves over the Natural method as they share complexities but the former prunes more heavily the search tree. More complex models, such as Recompute-BigM and Lagrangian, can visit up to five orders of magnitude less nodes than the existing method; this ratio increasing rapidly with the size of the matrices.

These results must be compared to the mean runtime ratio, as shown in the table. While the Lagrangian method variants provide the best pruning, its running time increases more rapidly. Experiments thus tend to show that Fixed-BigM and Recompute-BigM are generally the best choices.

The various parameters tested for the Lagrangian shows that using fewer gradient descent iterations worsen the computed upper bounds and thus increases the number of visited nodes, but it is counterweighted by the decrease in computation time, leading to very small differences in running time. Skipping multipliers updates does not seem to help, either.

8.2. Large neighborhood search on bigger instances

We now show the results on bigger matrices. These synthetic instances are filled by a Gaussian noise of parameters $(\mu = -0.01, \sigma = 1)$. We then inject a submatrix with Gaussian noise $(\mu = 0.01, \sigma = 1)$, the size of the submatrix depending on a parameter p being the ratio of the main matrix being filled with the submatrix. We generated 30 matrices for each (size, p) pair. All methods run a Large Neighborhood Search (LNS, (Shaw, 1998)) with an adaptive relaxation rate. The additional constraint (maximum running time and timeout per LNS iteration) and results are shown in Table 5 and Figure 3, which shows the average solution quality at any point in time for all the different methods. The average solution quality is defined as the mean of the ratios between the current solution for a method/instance and the best seen solution at timeout for all the methods.

In practice, the quality of the solution is more dependent on the diversifica-

Table 4: Average ratio between the number of visited nodes by *Natural* and the number of nodes visited by the other methods, along with the time ratio, on square matrices filled with a $\mathcal{N}(\cdot, 1)$ noise. Fifty different matrices per size. Static branching.

		Size - $\mathcal{N}(0,1)$						Size - $\mathcal{N}(0.2,1)$					
Method		10	14	18	22	26	30	10	14	18	22	26	30
Natural (Base for ratios)	Avg. visited nodes	83	689	8709	82k	918k	10461k	77	613	6k	48k	416k	3410k
	Time (s)	0.02s	0.04s	0.15s	1.01s	14.0s	162s	0.02s	0.04s	0.12s	0.51s	3.56s	40.8s
Natural+LB+fast	Node ratio	0.9	0.59	0.57	0.51	0.46	0.48	2.96	2.18	3.0	2.75	2.49	3.09
	Time ratio	1.18	1.08	1.09	0.84	1.02	1.12	1.49	1.49	1.92	1.95	2.08	3.49
Natural+LB	Node ratio	2.18	1.44	1.36	1.26	1.19	1.2	6.1	4.1	5.4	4.75	4.4	5.41
	Time ratio	1.19	1.09	1.04	0.81	0.92	0.93	1.47	1.45	1.71	1.68	1.72	2.57
Fixed-BigM	Node ratio	2.39	2.2	2.27	2.03	2.03	2.5	9.71	13.75	27.94	38.67	50.39	104.32
	Time ratio	1.17	1.12	1.32	1.19	1.24	1.69	1.43	1.63	2.67	4.39	7.52	17.87
Recompute-BigM	Node ratio	6.36	11.7	25.17	34.03	51.58	96.21	12.78	35.35	117.94	282.13	676.64	2033.6
	Time ratio	**1.25**	**1.21**	**1.62**	**2.73**	**4.29**	**9.43**	**1.55**	**1.73**	**2.74**	**5.37**	**13.33**	**48.52**
Lagrangian (50, all)	Node ratio	5.77	12.34	31.08	46.67	72.07	136.16	12.56	38.42	182.74	825.02	3490.77	20746.94
	Time ratio	0.56	0.22	0.17	0.17	0.27	0.4	0.71	0.47	0.38	0.72	2.12	12.91
Lagrangian (100, all)	Node ratio	5.99	14.78	46.97	83.3	140.27	291.93	12.65	39.74	206.35	1027.77	5387.54	35679.89
	Time ratio	0.36	0.13	0.15	0.18	0.31	0.51	0.47	0.25	0.28	0.62	1.95	12.12
Lagrangian (150, all)	Node ratio	6.0	14.94	48.0	86.29	145.97	306.46	12.65	40.2	209.15	1036.1	5506.0	36598.44
	Time ratio	0.23	0.1	0.13	0.15	0.25	0.4	0.35	0.16	0.22	0.51	1.6	10.05
Lagrangian (150, skip 30%)	Node ratio	5.85	14.58	46.41	82.82	140.57	294.4	12.62	39.52	205.0	1005.77	5392.64	35309.34
	Time ratio	0.35	0.13	0.14	0.18	0.32	0.53	0.49	0.28	0.25	0.65	2.03	12.72
Lagrangian (150, skip 60%)	Node ratio	5.26	13.03	40.9	70.18	113.99	233.59	12.48	38.0	193.77	954.92	4839.97	31373.44
	Time ratio	0.6	0.28	0.19	0.23	0.45	0.76	0.81	0.6	0.53	0.95	2.54	14.27
Lagrangian (150, skip 90%)	Node ratio	4.74	8.07	18.42	27.98	43.04	83.36	12.29	34.81	142.72	570.75	1966.95	10727.53
	Time ratio	1.14	0.68	0.36	0.39	0.78	1.21	1.44	1.63	2.19	3.05	5.94	25.67
Lagrangian (150, skip 95%)	Node ratio	4.71	6.72	14.09	19.46	31.43	59.62	12.26	34.44	131.05	461.54	1277.86	5412.74
	Time ratio	1.15	0.73	0.38	0.45	0.9	1.46	1.44	1.66	2.04	2.91	5.03	23.58

Table 5: Solution quality (average ratio of objective value found by a method divided by the best objective found by any method) using Large Neighborhood Search. All numbers are in percentages. Best methods (less than 0.05% wrt the best method) are in bold. Average of 30 different instances per type of instance.

Matrix size	1000			2000			3000		
p	5%	30%	50%	5%	30%	50%	5%	30%	50%
CP-based									
Natural	99.31	99.17	99.53	99.2	99.3	99.55	98.54	98.67	99.38
Natural+LB+fast	**99.5**	99.39	**99.6**	**99.49**	**99.63**	**99.81**	99.49	99.57	99.75
Natural+LB	99.44	99.42	**99.6**	**99.45**	99.49	99.64	**99.63**	99.46	**99.85**
Fixed-BigM	99.39	**99.51**	99.54	**99.46**	99.49	99.61	99.51	**99.66**	99.79
Recompute-BigM	99.37	99.4	**99.56**	**99.5**	99.51	**99.77**	99.55	99.57	99.73
Lagrangian (50, all)	99.13	99.29	99.46	98.18	98.49	99.04	93.74	94.45	96.9
Lagrangian (100, all)	99.3	99.27	99.42	97.73	98.08	98.78	91.12	91.07	94.84
Lagrangian (150, all)	99.24	99.2	99.42	97.22	97.59	98.45	88.61	88.86	92.49
Lagrangian (150, skip 30%)	99.2	99.29	99.44	97.89	98.24	98.8	92.54	93.59	96.33
Lagrangian (150, skip 60%)	99.23	99.27	99.53	98.65	98.64	99.19	95.32	95.89	97.87
Lagrangian (150, skip 90%)	99.29	99.26	**99.57**	99.3	99.09	99.4	97.36	97.62	98.78
Lagrangian (150, skip 95%)	99.27	99.45	**99.59**	99.1	99.15	99.42	97.31	97.68	98.8
MIP-BigM	85.06	85.71	89.41	83.07	84.71	88.95	80.19	82.03	89.16
MIP-Strong*	0.0	0.0	2.98	0.0	0.0	2.22	OOM	OOM	OOM

* in practice, MIP-Strong cannot solve the first LP relaxation in 60 seconds on most instances.

tion method[3] than on the intensification method (the bound used). The new bounds produce only a small difference in terms of results after 60 seconds, as seen in Table 5. Figure 3 shows, however, that the convergence rate is higher for some methods. The group of methods that performs better can be summarized as follows:

[3]here, a part of the best current solution is randomly kept, and the remaining columns are aggregated until the matrix is small enough, these parameters being computed at runtime

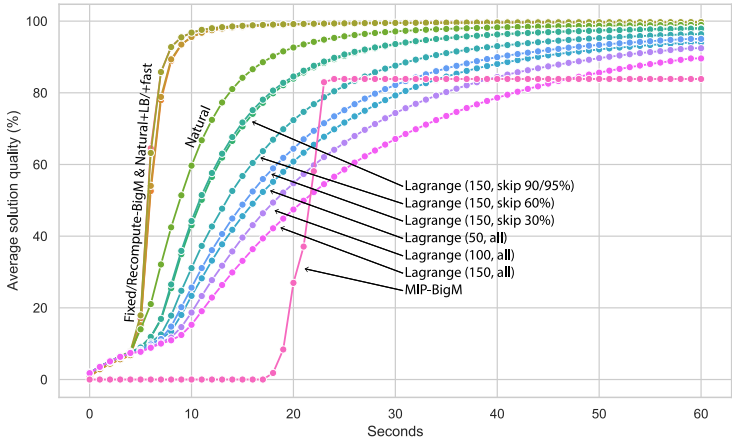

Figure 3: Average solution quality w.r.t. time for all methods. (MIP-Strong is ommited due to poor performance)

Table 6: Average of ratios between the UB found by the upper bounding procedure and the best found solution, on 50 instances for each type of instance. Lower is better.

	n=50, $\mathcal{N}(0,1)$	n=50, $\mathcal{N}(0.2,1)$	n=100, $\mathcal{N}(0,1)$	n=100, $\mathcal{N}(0.2,1)$
Natural	2.82	1.62	6.03	2.14
Natural+LB	2.83	1.63	6.03	2.14
Natural+LB+Fast	2.70	1.59	5.85	2.09
Fixed-BigM	1.82	1.16	3.38	1.38
Recompute-BigM	1.71	1.04	3.39	1.35
Lagrange (50)	1.93	1.05	3.50	1.22
Lagrange (100)	1.85	1.02	3.36	1.16
Lagrange (150)	1.86	1.02	3.37	1.16

- They all use the lower bound for pruning the domain;
- They use a moderate amount of computational time (i.e. they are not based on a gradient descent).

The gradient descent needed to run the Lagrangian-based methods appears too costly.

8.3. Upper bounding

We use the new bounds to attempt to close the gap between the incumbent solution (which is effectively a lower bound for the optimum) and the upper bound. To this end, we use the following algorithm, based on a priority queue which always has as first item the node with the highest UB:

1. Dequeue the node
2. Expand it using the static branching presented earlier, creating two new nodes
3. Put the two new nodes in the priority queue.

We let this algorithm run for a 600 seconds, with the various bounds, on various relatively large matrices. We provide a lower bound to the algorithm (enhancing its pruning ability) which is the best found solution by the Fixed-BigM method after 300 seconds.

The results are presented in Table 6, which shows the average ratio between the best found solution and the UB obtained by the algorithm above.

Despite their slowness, the Lagrangian-based method provides the best bounds for simpler instances (filled with a $\mathcal{N}(0.2,1)$), while for instances filled with $\mathcal{N}(0,1)$ the methods based on the BigM formulation work best.

8.4. Real-life data

Table 7 shows runtime, reached objective value, and runtime at which the best objective value was reached, from multiple real-life datasets (Le Van et al., 2014; Branders et al., 2019a).

Table 7: Results of a complete search on real-life data. All methods had 1 hour of maximum runtime.

Method	Best solution	(time)	Total runtime	Total #nodes	Optimum proven
`bc_030.tsv` (2211×94)					
Natural	4052763	(2.5s)	1h (TO)	1099464	✗
Natural+LB	4052763	(1.2s)	1h (TO)	599436	✗
Natural+LB+Fast	4052763	(1.5s)	26m	1095513	✓
Fixed-BigM	4052763	(0.7s)	2s	95	✓
Recompute-BigM	4052763	(1s)	1.6s	25	✓
Lagrange (50, all)	4052763	(45s)	61s	23	✓
Lagrange (100, all)	4052763	(83s)	108s	23	✓
Lagrange (150, all)	4052763	(121s)	160s	23	✓
Lagrange (150, skip 30%)	4052763	(73s)	100s	23	✓
Lagrange (150, skip 60%)	4052763	(35s)	47s	23	✓
Lagrange (150, skip 90%)	4052763	(3.7s)	8.1s	33	✓
Lagrange (150, skip 95%)	4052763	(3.2s)	7.2s	33	✓
MIP-BigM	4052763	(0.0s)	1.0s		✓
MIP-Strong	OOM		OOM		✗
`ijcai_small_0.075.tsv` (100×100)					
Natural	209.35	(0.1s)	106s	742703	✓
Natural+LB	209.35	(0.07s)	110s	742703	✓
Natural+LB+Fast	209.35	(0.06s)	62s	5124739	✓
Fixed-BigM	209.35	(0.06s)	121s	5100199	✓
Recompute-BigM	209.35	(0.1s)	367s	733563	✓
Lagrange (50, all)	209.35	(1.2s)	1h (TO)	65457	✗
Lagrange (100, all)	209.35	(1.7s)	1h (TO)	7921	✗
Lagrange (150, all)	209.35	(3.1s)	1033s	7873	✓
Lagrange (150, skip 30%)	209.35	(2.1s)	776s	8217	✓
Lagrange (150, skip 60%)	209.35	(0.7s)	531s	9889	✓
Lagrange (150, skip 90%)	209.35	(0.2s)	485s	51365	✓
Lagrange (150, skip 95%)	209.35	(0.2s)	608s	114299	✓
MIP-BigM	209.35	(0.03s)	1h (TO)		✗
MIP-Strong	209.35	(355s)	1226s		✓
`olympic_0.02.tsv` (74×151)					
Natural	28.955	(0.16s)	1h (TO)	259652006	✗
Natural+LB	29.128	(0.07s)	953s	30626451	✓
Natural+LB+Fast	29.128	(0.04s)	556s	41019839	✓
Fixed-BigM	29.128	(0.07s)	475s	11472295	✓
Recompute-BigM	29.128	(0.09s)	13s	17241	✓
Lagrange (50, all)	29.128	(3.8s)	1h (TO)	14658	✗
Lagrange (100, all)	29.128	(7.1s)	29s	163	✓
Lagrange (150, all)	29.128	(8.9s)	10s	23	✓
Lagrange (150, skip 30%)	29.128	(6.2s)	8.2s	23	✓
Lagrange (150, skip 60%)	29.128	(4.0s)	5.9s	25	✓
Lagrange (150, skip 90%)	29.128	(1.0s)	2.3s	27	✓
Lagrange (150, skip 95%)	29.128	(1.0s)	3.2s	171	✓
MIP-BigM	29.128	(0.03s)	0.44s		✓
MIP-Strong	29.128	(1.56s)	1.56s		✓

The result on these medium-sized instance, combined with the ones presented in the previous sections, shows that there is no silver bullet.

On `bc_030.tsv` (breast cancer dataset (Le Van et al., 2014)), the best methods are based on the BigM formulation (fixed, recompute, and MIP), visiting significantly fewer nodes. Lagrangian methods are penalized due to their complexity on this matrix with 2211 rows.

On `ijcai_small_0.075.tsv` (graph of the 100 most prolific authors at 2019's IJCAI and the 100 venues to which they publish the most), the BigM and Lagrangian formulations seem to give bound marginally equivalent to the natural bound, and so are not able to perform well, with the MIP even reaching timeout.

On `olympic_0.02.tsv` (Number of medals won per sport and per country in all olympic games, (Branders et al., 2019a)) we observe a significant difference (3 orders of magnitude) between Fixed-BigM and Recompute-BigM, and again 3 orders of magnitude with Lagrangian-based methods, which prove useful on this particular instance.

9. Conclusion

We presented three new methods for computing upper bounds for the Maximum-Sum Submatrix (MSS) problem. The first two methods (Fixed-BigM and Recompute-BigM) are based on a Big M reformulation of the problem; Fixed-BigM is shown to be computable in linear time (depending on the number of rows or columns), while Recompute-BigM, which is more tight, can be computed in quadratic time (depending on the number of cells in the matrix). A third method, based on a Lagragian relaxation of the MSS, is also presented.

We show the Lagrangian method gives the best bound, both theoretically and experimentally, but suffers from its complexity (quadratic, with an additional gradient descent). Fixed-BigM and Recompute-BigM are shown to produce tighter bounds, as they provide a middle ground between more tight but slower Lagrangian-based methods and the less precise but faster natural bound.

Source code and raw experiment results

The source code is available on Zenodo(Derval & Schaus, 2020a), along with the experiments code and results (Derval & Schaus, 2020b). The experiment were run on the OscaR CP solver (OscaR Team, 2012).

References

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In *Advances in knowledge discovery and data mining* (pp. 307–328). USA: American Association for Artificial Intelligence.

Boyd, S., Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Branders, V., Derval, G., Schaus, P., & Dupont, P. (2019a). Mining a Maximum Weighted Set of Disjoint Submatrices. In P. Kralj Novak, T. muc, & S. Deroski (Eds.), *Discovery Science* Lecture Notes in Computer Science (pp. 18–28). Cham: Springer International Publishing. doi:10.1007/978-3-030-33778-0_2.

Branders, V., Schaus, P., & Dupont, P. (2017). Mining a sub-matrix of maximal sum. In *Proceedings of the 6th International Workshop on New Frontiers in Mining Complex Patterns in conjunction with ECML-PKDD 2017*.

Branders, V., Schaus, P., & Dupont, P. (2019b). Identifying gene-specific subgroups: an alternative to biclustering. *BMC Bioinformatics*, *20*, 625. URL: https://doi.org/10.1186/s12859-019-3289-0. doi:10.1186/s12859-019-3289-0.

Cambazard, H., & Fages, J.-G. (2015). New filtering for AtMostNValue and its weighted variant: A Lagrangian approach. *Constraints*, *20*, 362–380. URL: http://link.springer.com/10.1007/s10601-015-9191-0. doi:10.1007/s10601-015-9191-0.

Dao, T. H., Docquier, F., Maurel, M., & Schaus, P. (2018). Global Migration in the 20th and 21st Centuries: the Unstoppable Force of Demography. URL: https://hal.archives-ouvertes.fr/hal-01743799 fERDI Working paper P223.

Derval, G., Branders, V., Dupont, P., & Schaus, P. (2019). The maximum weighted submatrix coverage problem: A cp approach. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 258–274). Springer.

Derval, G., & Schaus, P. (2020a). Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach: source code. URL: https://doi.org/10.5281/zenodo.3992317. doi:10.5281/zenodo.3992317.

Derval, G., & Schaus, P. (2020b). Software Open Access Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach: experiment code and results. URL: https://doi.org/10.5281/zenodo.3992324. doi:10.5281/zenodo.3992324.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2006). *Column Generation*. Springer Science & Business Media.

Focacci, F., Lodi, A., & Milano, M. (1999). Cost-based domain filtering. In *International Conference on Principles and Practice of Constraint Programming* (pp. 189–203). Springer.

Geerts, F., Goethals, B., & Mielikinen, T. (2004). Tiling Databases. In E. Suzuki, & S. Arikawa (Eds.), *Discovery Science* Lecture Notes in Computer Science (pp. 278–289). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-30214-8_22.

Griva, I., Nash, S. G., & Sofer, A. (2009). *Linear and nonlinear optimization* volume 108. Siam.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, *8*, 53–87. URL: https://doi.org/10.1023/B:DAMI.0000005258.31418.83. doi:10.1023/B:DAMI.0000005258.31418.83.

Hartigan, J. A. (1972). Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, *67*, 123–129. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1972.10481214. doi:10.1080/01621459.1972.10481214.

Le Van, T., van Leeuwen, M., Nijssen, S., Fierro, A. C., Marchal, K., & De Raedt, L. (2014). Ranked Tiling. In T. Calders, F. Esposito, E. Hllermeier, & R. Meo (Eds.), *Machine Learning and Knowledge Discovery in Databases* Lecture Notes in Computer Science (pp. 98–113). Berlin, Heidelberg: Springer. doi:10.1007/978-3-662-44851-9_7.

Madeira, S. C., & Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, *1*, 24–45.

Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., & Diot, C. (2002). Traffic matrix estimation: Existing techniques and new directions. In *ACM SIGCOMM Computer Communication Review* (pp. 161–174). ACM volume 32.

OscaR Team (2012). OscaR: Scala in OR. Available from https://bitbucket.org/oscarlib/oscar.

Peeters, R. (2003). The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, *131*, 651–654. URL: http://www.sciencedirect.com/science/article/pii/S0166218X03003330. doi:10.1016/S0166-218X(03)00333-0.

Pontes, B., Girldez, R., & Aguilar-Ruiz, J. S. (2015). Biclustering on expression data: A review. *Journal of Biomedical Informatics*, *57*, 163–180. URL: http://www.sciencedirect.com/science/article/pii/S1532046415001380. doi:10.1016/j.jbi.2015.06.028.

Punnen, A. P., Sripratak, P., & Karapetyan, D. (2015). The bipartite unconstrained 01 quadratic programming problem: Polynomially solvable cases. *Discrete Applied Mathematics*, *193*, 1–10. URL: http://www.sciencedirect.com/science/article/pii/S0166218X15001742. doi:10.1016/j.dam.2015.04.004.

Rother, C., Kolmogorov, V., Lempitsky, V., & Szummer, M. (2007). Optimizing Binary MRFs via Extended Roof Duality. In *2007 IEEE Conference on*

Computer Vision and Pattern Recognition (pp. 1–8). Minneapolis, MN, USA: IEEE. URL: http://ieeexplore.ieee.org/document/4270228/. doi:10.1109/CVPR.2007.383203.

Schaus, P., Aoga, J. O. R., & Guns, T. (2017). CoverSize: A Global Constraint for Frequency-Based Itemset Mining. In J. C. Beck (Ed.), *Principles and Practice of Constraint Programming* Lecture Notes in Computer Science (pp. 529–546). Cham: Springer International Publishing. doi:10.1007/978-3-319-66158-2_34.

Sellmann, M. (2004). Theoretical Foundations of CP-Based Lagrangian Relaxation. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming CP 2004* Lecture Notes in Computer Science (pp. 634–647). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-30201-8_46.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming* (pp. 417–431). Springer.

Tan, J. (2008). Inapproximability of Maximum Weighted Edge Biclique and Its Applications. In M. Agrawal, D. Du, Z. Duan, & A. Li (Eds.), *Theory and Applications of Models of Computation* Lecture Notes in Computer Science (pp. 282–293). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-79228-4_25.

Tanay, A., Sharan, R., & Shamir, R. (2002). Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, *18*, S136–S144. URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/18.suppl_1.S136. doi:10.1093/bioinformatics/18.suppl_1.S136.

The World Bank (2018). Migration and remittances data. Data retrieved from "Bilateral Migration Matrix 2018", http://www.worldbank.org/en/topic/migrationremittancesdiasporaissues/brief/migration-remittances-data.

Van't Veer, L. J., Dai, H., Van De Vijver, M. J., He, Y. D., Hart, A. A., Mao, M., Peterse, H. L., Van Der Kooy, K., Marton, M. J., Witteveen, A. T. et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *nature*, *415*, 530.

Xie, J., Ma, A., Fennell, A., Ma, Q., & Zhao, J. (2018). It is time to apply biclustering: a comprehensive review of biclustering applications in biological and biomedical data. *Briefings in Bioinformatics*, *20*, 1450–1465. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6931057/. doi:10.1093/bib/bby014.