Finding maximum sum submatrices

Guillaume Derval

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

December 2021

ICTEAM
Louvain School of Engineering
UCLouvain
Louvain-la-Neuve
Belgium

Finding maximum sum submatrices
by Guillaume Derval

ICTEAM
UCLouvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

Preamble

Data mining is nowadays an important part of numerous fields of science, from linguistics [Ped99] to ecology [Han+20], via chemistry [TZR18] and medicine [LZ05]. Since the beginning of this century, many multi-billion-dollar companies, that focus on data mining as it represents their main source of income, have emerged. Data mining is now pervasive in our society. The field is thus very active, and numerous results, techniques, studies have been made to solve problems encountered in many different disciplines.

Among all these problems, this thesis focuses on a particular family, biclustering. As its name indicates, this ensemble of methods focuses on finding *biclusters*, clusters of two different kinds of entities. Generally, these problems are defined on matrices, where the biclusters are actually a selection of rows and columns, forming a submatrix (the bicluster). Biclustering has been applied widely: movie recommendation [UF98], text mining [Dhi01] and gene expression data analysis [MO04] are examples among many others.

In this thesis we explore a specific problem of finding maximum-sum submatrices in matrix data. This problem emerged in the context of the analysis of gene expression data [BSD17a], matrices where each column represents a *gene* and each row a *sample* (a *condition*, a *patient*, a *tissue* or any other entity where gene can express themselves); a cell in such a matrix represents the expression of a gene on an entity, which can be measured in multiple ways. A submatrix with a large sum can represent a group of patients that share a common illness, induced by a group of genes, for example. In practice, research has shown that such techniques are biologically sound [BSD19]. We focus on the mathematical, algorithmic and optimization aspects of these problems in this thesis. For an extensive discussion of the biological aspects, see [Bra21].

Maximum-sum submatrices are more than a tool for analyzing gene expression data; in general, they can be used in any context where there are relations of various intensities between two distinct groups. As a data mining tool, they can extract previously unknown information from this family of datasets, in an unsupervised or semi-supervised way (by modifying the initial matrix interactively or adding additional constraints). Another example of such a dataset is bilateral migration flows [Dao+18a].

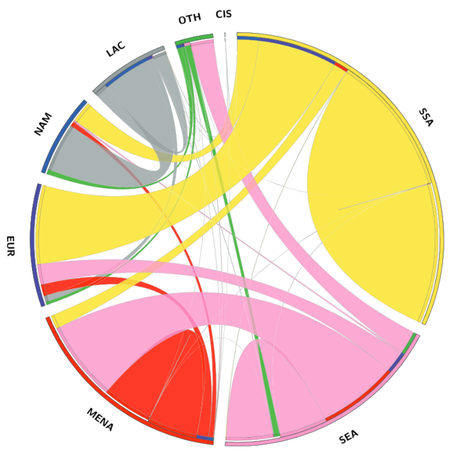


Figure 1: A Circos plot representing a migration matrix, with flows being cells of a country/country matrix. Reproduced with authorization from [Dao+18a].

After a chapter dedicated to an overview of the existing methods, problems and techniques used in this thesis, Chapter 2 of this manuscript focuses on finding the submatrix of maximum sum (Maximum-sum submatrix problem, or MSS), or at least a good approximation; for this we use linear and Lagrangian relaxations of the problem to infer fast-to-compute upper bounds for the problem and compute reduced-costs, enabling a reduction of the (exponential) number of possible submatrices to consider.

An important limitation of the approach of finding the best maximum-sum submatrix is its poor configurability; simply plugging matrices into an MSS solver may give very large matrices or empty matrices if the data is not balanced correctly. In chapter 3 we add cardinality constraints on the size (number of rows and columns) of the found submatrices, in order to allow the users to indicate what interests them, and to improve interpretability compared to other techniques also described in this chapter.

In the second part of this thesis, we look into ways of finding multiple interesting submatrices. Chapter 4 discusses techniques for mining overlapping submatrices. We first note that we must penalize the overlaps and describe precisely the problem being tackled, then use a constraint programming approach with dominance rules and upper bound computations based on finite-state machines, along with a large neighborhood search (LNS).

Chapter 5 tackles the Maximum Weighted Set of Disjoint Submatrices (MWSDS) problem, which doesn't allow overlaps. We use column generation to solve this problem, which leads to an interesting usage of the results gathered on the MSS problem.

In the last chapter, chapter 6, we discuss further research directions.

Bibliographic notes

This thesis led to the publication of, and is based on, the following works:

1. G. Derval, V. Branders, P. Dupont, and P. Schaus. "The Maximum Weighted Submatrix Coverage Problem: A CP Approach". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by L.-M. Rousseau and K. Stergiou. Cham: Springer International Publishing, 2019, pp. 258–274. ISBN: 978-3-030-19212-9.

2. V. Branders, G. Derval, P. Schaus, and P. Dupont. "Mining a maximum weighted set of disjoint submatrices". In: *International Conference on Discovery Science*. Springer. 2019, pp. 18–28.

3. G. Derval and P. Schaus. "Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach". In: *European Journal of Operational Research* (2021). ISSN: 0377-2217. DOI: `https://doi.org/10.1016/j.ejor.2021.06.008`. URL: `https://www.sciencedirect.com/science/article/pii/S0377221721005142`.

During the five years that were needed to write this thesis, other unrelated but still interesting works were published by the author:

- G. Derval, J.-C. Régin, and P. Schaus. "Improved filtering for the bin-packing with cardinality constraint". In: *Constraints* 23.3 (2018), pp. 251–271.

- G. Derval, F. Docquier, and P. Schaus. "An aggregate learning approach for interpretable semi-supervised population prediction and disaggregation using ancillary data". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 672–687.

Acknowledgments

In the years spent writing this thesis (among other things), and in general during my studies at UCLouvain, I met many persons with whom I spend ten wonderful and stimulant years.

I knew many of them as fellow students. The moments we shared in the Intel Room, working on projects, discussing politics and joking will be remembered as some of the best of my life. Anthony, Gorby, Mathieu, Charles, Thibault, Benoit(s), thanks.

I had the opportunity to work as a student sysadmin administrator, under the supervision of Pierre Reinbold and Nicolas Detienne; I enjoyed the time spent with Anthony Gégo, François Michel and Maxime Piraux fixing the always broken and outdated Scientific Linux 6 (or later CentOS 7) distributions in the student computer rooms. I learned many skills, and I can't thank them all enough for that. The INGInious project, started with the impulse of Pierre and Olivier Bonaventure, allowed Anthony and I, I think and hope, making a small but significative contribution to the way computer science is taught and learned nowadays at UCLouvain. That would not have been possible without the guidance, help, support and work from Anthony, Pierre and Olivier.

The choice of specializing in AI and (combinatorial) optimization was greatly inspired by the courses and interactions I had with Yves Deville, Pierre Dupont and Pierre Schaus, who later became my advisor, first for my master's thesis then for my PhD. The discussion I had with them, along with my fellow PhD students in the BeCool team, Hélène Verhaeghe, Sascha Van Cauwelaert, Alexandre Dubray, Charles Thomas, Xavier Gillard, John Aoga, Alex Mattenet and others I sadly forgot to put here were (and still are) stimulating, insightful, and interesting, whether they were about our respective research or not. We spent together memorable times in many conferences! Sascha in particular spent his final PhD years in my office (or rather, I spent my first PhD years in his office), and I'll always remember the fun I had to listen to Radio Paradize with Sascha singing, with French lyrics, English songs.

The other members of the department are not forgotten either, particularly all the persons regularly present at the cafeteria. The conversations we shared, sometimes about research, sometimes about news, sometimes about politics, and more often than not on politically incorrect subjects

were very fun times.

I'd like to specifically thank the administrative and technical personnel of the INGI department. Vanessa and Sophie, which I bothered quite a lot of times with strange questions and requests; I hope I helped them enough in return! The sysadmins, first Pierre and Nicolas, then Nicolas, Anthony and Ludovic, were also an invaluable help during this thesis, providing needed tools and fixing things I broke.

Pierre Schaus, as written a bit everywhere in this thesis, was my advisor during these 5 years. I cannot count the number of discussions we had about research, but also about lots of other subjects. His guidance was truly fantastic, always positive and focus on pushing me in the right direction. I feel very lucky to have pursued a PhD with him as advisor.

The last two years (2020-2021) were marked by the COVID-19 pandemic, and I happened to spend quite some time analyzing data, along with a group of people known as CovidRationnel. Some of our discussions were truly eye opening, and the opportunity for me to work again with so many scientists from so many different fields will probably not happen anytime soon.

But first and foremost, this work could not have been possible without the indefectible support from my family. Particularly from my mother and my grandparents, who always pushed me to do better. And from my brothers and sisters, Mathurin, Arthur, Guillaume and Céleste. These last 10 years, I've shared my life with Charlotte, the most thoughtful and attentive person I've ever met. She had the strength to support my (luckily not so numerous) days of demotivation, and my more-often-than-not grumpy attitude. She gave birth to Ondine in 2020, who is the other sun in my life since, and to whom I dedicate this thesis[1].

Other Acknowledgments

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

[1]Even though she did not help to hand it in time! :-)

Contents

Background | 1

This chapter provides an overview of most of the concepts and methods needed to understand the work made in this thesis (each time with references to books for further reading), along with existing works related to the subject of this thesis.

1.1 Similar problems

1.1.1 Biclustering

Pattern mining is a well-known and widely studied field of research because of the numerous real-life applications (health, DNA analysis, errors tracking and detectors, marketing ...)

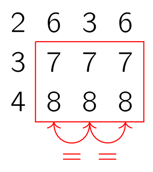
This section provides an overview on Biclustering problems and algorithms. We redirect to the very complete survey by Madeira and Oliveira [MO04] for more precisions.

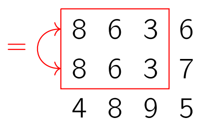
The term *biclustering* represents a family of similar problem that attempts to extract from matrices some kind of *biclusters* (clusters of rows and columns). This kind of clustering arises particularly in the context of gene expression data: matrices gathering the expression level of multiple genes (represented as columns) in multiple organisms (rows). This definition is voluntarily vague, as *biclustering* covers a family of methods. Madeira and Oliveira [MO04] separates these into four families, depending on the type of biclusters that are being searched for, themselves separated into subfamilies (with examples of such methods each time):

- Biclusters with constant values [Har72a]

$$\begin{array}{|ccc|c}
\cline{1-3}
8 & 8 & 8 & 6 \\
8 & 8 & 8 & 7 \\
\cline{1-3}
\multicolumn{1}{c}{4} & 8 & \multicolumn{1}{c}{9} & 5
\end{array}$$

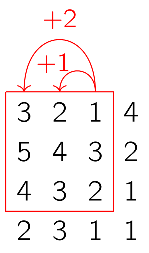
- Biclusters with constant values on rows or columns [CST+00; SMD03]

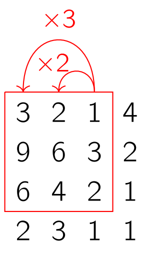




Constant rows Constant columns

- Biclusters with coherent values

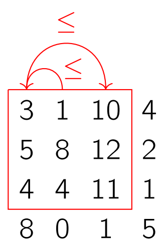




Additive model [CC00]
(each row = another + constant)

Multiplicative model [GLD00]
(each row = another × constant)

- Biclusters with coherent evolutions



Order-preserving submatrix (OPSM) [Ben+02]

In an OPSM, each column is interpreted as an ordering, and each ordering in the OPSM must be the same. In this example, given the matrix is named A, we have in each column j that $a_{0,j} < a_{2,j} < a_{1,j}$.

- Other types of bicluster exists, such as the Maximum-Sum Submatrix problem. These focus less on the internal structure of the bicluster but rather on its overall value, evaluated by given metrics.

These families and subfamilies themselves encompass multiple formulations that handle noise in the data in different ways.

1.1.2 Frequent Itemset Mining

Frequent Itemset Mining [AH14] is a set of techniques aiming at finding a set (an *itemset*) of *items* present together in a large set of *transactions*. The stereotypical example of Frequent Itemset Mining is of finding large number of objects bought together in stores.

More formally, given a set of items $I = \{i_1, \ldots, i_n\}$, a database of transactions $D = \{T_1, \ldots, T_m\}$ with each $T_j \subseteq I$, and a minimum support θ, the goal is to find a set $P \subseteq I$ such that

$$\text{freq}(P) = |\{i \mid T_i \in D \land P \subseteq T_i\}| \geq \theta$$

that is, the number of transactions containing all the items in P is greater or equal than θ. Such an itemset P is said to be *frequent* (w.r.t. θ).

There are numerous alternative formulations of problems in this context. Among the ones with the most extensive literature, we have:

- Maximal itemset mining: a frequent itemset is said to be maximal if no superset of the itemset is frequent. Maximal itemset mining thus aims at enumerating all maximal itemsets.
- Closed itemset mining: an itemset is said to be closed if no supersets have the same support (number of transactions including the itemset).

While less used in practice than the "transaction list" view, a matrix formulation of the problem exists. Given the transaction database D and the set of items I, we can construct a binary matrix $\in \mathbb{R}^{m \times n}$, such that each column represent an item, and a row a transaction. The content of a cell is 1 if the item is present in the transaction or 0 otherwise. Finding a frequent itemset of a given frequency is thus equivalent to finding a non-contiguous submatrix filled only with ones in such a matrix, with at least θ rows.

An important property largely used in the field of frequent itemset mining is that frequency is antimonotonic: if an itemset is not frequent, any supersets of the itemset won't be frequent either. This property, called in the field the *apriori principle*[AS+94], is used in most algorithms to heavily prune the set of candidates.

1.1.3 Maximum-edge biclique problem

The maximum-edge biclique problem (MBP)[Pee03; GG08; GG13] is to find complete (unweighted) bicliques in bipartite graphs, and more precisely the ones with the largest number of edges. The link to the problems discussed in this thesis is again made with a matrix representation: given a bipartite graph $G =< V, E >$ with two groups of nodes A and B ($A \bigcap B = \emptyset$, $A \bigcup B = V$, $E \subseteq A \times B$), an adjacency matrix can be built with a row for each node in A and a column for each node in B, each cell containing 1 if there is an edge between the nodes it represents, 0 otherwise.

The setting is thus very similar to the one in Frequent Pattern Mining: MBP aims at finding large non-contiguous submatrices filled only with ones. The main difference is the objective function to be maximized, here the number of edges in the resulting biclique, or said differently the area of the submatrix found. The decision version of this problem (is there a biclique with at least k edges, or is there is submatrix filled only with ones with an area of at least k?) is NP-complete[Pee03].

As in FIM, there are variants for mining maximal bicliques (ones not contained in a larger biclique) rather than only finding the largest one (the maximum).

1.1.4 Contiguous maximum-sum submatrix

We focus in this thesis on finding non-contiguous submatrices of maximum sum. The contiguous case, where the selected rows and columns must actually be one next to another in the submatrix, is a problem well known to be solvable in polynomial time, and is called the two-dimensional maximum *subarray* problem.

Indeed, for a $m \times n$ matrix, there are $\mathcal{O}(2^{m+n})$ possible non-contiguous submatrices, but only $\mathcal{O}(m^2n^2)$ subarray, which can be enumerated simply by enumerating all pairs of cells. As computing the sum of a submatrix is trivially $\mathcal{O}(mn)$ at most, we have a trivial algorithm in $\mathcal{O}(m^3n^3)$ to solve this problem, as shown in Algorithm 1.

An alternative algorithm is to use Kadane's algorithm for 1d maximum-sum subarray. This algorithm solves in $\mathcal{O}(n)$ the one-dimensional case by iterating on the array and maintaining the best sum ending at the current cell.

Using Kadane's algorithm as a subroutine, we can:

- Produce, for each pair of columns, an array of size m where each cell i contains the sum of the content of the row i between the two columns. This operation is done in $\mathcal{O}(n^2m)$ by reusing previous computations.

- Use Kadane's algorithm on the resulting arrays to find the best sub-array, providing two rows, which, in addition to the two columns on which we iterate, gives the best sum submatrix for this choice of

Algorithm 1 Naive algorithm for finding a maximum-sum subarray in a $m \times n$ matrix

```
($M \in \mathbb{R}^{m \times n}$ is a matrix)
bestSum ← 0                                            ▷ Best sum seen so far
best ← (0, 0, 0, 0)               ▷ Coordinates of the best submatrix seen so far

for all row $a \in \{1, \dots, m\}$ do                   ▷ Row where the submatrix starts
    for all row $b \in \{a+1, \dots, m+1\}$ do             ▷ Row where the submatrix ends
        for all column $x \in \{1, \dots, n\}$ do         ▷ Column where the submatrix starts
            for all column $y \in \{x+1, \dots, n+1\}$ do  ▷ Column where the submatrix ends
                $s \leftarrow 0$                                  ▷ Sum of the submatrix
                for all row $c \in \{a, \dots b-1\}$ do
                    for all column $z \in \{x, \dots y-1\}$ do
                        $s \leftarrow s + \mathrm{M}_{cz}$
                if $s >$ bestSum then
                    bestSum ← $s$
                    best ← $(a, b, x, y)$
return best
```

columns. This operation runs in $\mathcal{O}(m)$ for each of the n^2 arrays, so $\mathcal{O}(n^2 m)$ in total.

- The best submatrix seen during the computation is the best one overall.

This algorithm is presented in Algorithm 2 (for simplicity, we do not show how to recover the coordinates of the submatrix). This complexity cannot reliably be beaten as there exists a hardness result on square $n \times n$ matrices: the 2d maximum sum subarray problem is $\Omega(n^3)$ [BDT16].

Algorithm 2 $\mathcal{O}(n^2 m)$ algorithm for finding the maximum-sum subarray value of a 2d $m \times n$ array

```
function kadane(array $a$ of size $m$)
    best ← 0
    cur ← 0
    for $i \in \{1, \dots, m\}$ do
        cur ← max(0, cur + $a_i$)
        best ← max(best, cur)
    return best

best ← 0
for all column $x \in \{1, \dots, n\}$ do
    rsum ← array of size $m$ filled with zeros
    for all column $y \in \{x, \dots, n\}$ do
        for all row $i \in \{1, \dots, m\}$ do
            $\mathrm{rsum}_i \leftarrow \mathrm{rsum}_i + M_{iy}$
        best ← max(kadane(rsum), best)
return best
```

1.2 Constraint Programming

Constraint Programming (CP) is a *declarative* paradigm for solving *combinatorial satisfaction and optimization problems*. It primarily relies on *backtracking*, *heuristic search in search trees*, *branch-and-bound* and *constraint propagation*.

The specific class of problem being solved in CP are Constraint Satisfaction Problems (CSPs) and Constraint Optimization Problems (COPs):

Definition 1. *A* **Constraint Satisfaction Problem (CSP)***[RVW06] is a triple $\langle X, D, C\rangle$, where X is a set of variables $\{X_1, X_2, \ldots, X_n\}$ each having a non-empty domain, stored in the set $D = \{D_1, D_2, \ldots, D_n\}$. $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints.*

A constraint is defined as a relation between a subset of the variables of X (formally, its thus a pair $\langle s_i, R_i\rangle$ with $s_i \subseteq X$. s_i is also called the scope of the constraint). The relation is thus a subset of the Cartesian product of the domain of the variables in t_i.

A constraint is said to be satisfied by an assignment of the variables if the corresponding tuple is in the relation of the constraint.

A solution to a CSP is an assignment to all the variables that satisfies all the constraints.

If no solution exists (i.e. if the intersection of the relation is empty) the problem is said to be unsatisfiable. Otherwise it is satisfiable.

A variation exists for optimization problems[1]:

Definition 2. *A* **Constraint Optimization Problem (COP)** *is a CSP with an additional objective function, which gives for an assignment of all variables an objective value to be either maximized or minimized.*

In general in CP, the set of constraints available is very large (from equality constraints to "the graph represented by these variables must be a single cycle").

1.2.1 Inference and consistencies

One of the main ideas behind CP is the use of reasoning and inference to reduce the domains of variables. For this, it uses the concept of consistency. This term actually groups together a large number of techniques used to prune the domain from information known from the constraints. Here are some of the most well-known consistencies:

[1]As noted in [RVW06] there are multiple definitions for most of the concepts presented here; they are generally similar in practice although contradictory in theory. We try to stick to these definitions in the thesis.

Definition 3. *(Various consistencies)* **Arc consistency (AC)**. *W.r.t a binary constraint with a relation R and a scope $\langle X_1, X_2 \rangle$, the domain of these two variables is said be arc consistent if, for all value i in the domain of X_1, there exists a value j in the domain of X_2 such that $(i, j) \in R$, and conversely for each value of X_2.*

Generalized arc consistency (GAC) *is the extension of this principle to n-ary constraint, where all values in all domains must be supported by a tuple from the Cartesian product of the domains, with that specific value, that satisfies the constraint. GAC is also sometimes called* **domain consistency**.

Bound consistency *ensures that the minimum and maximum value in the domain of any variable is supported by a tuple taken in the Cartesian product of the domain while considering them a continuous.*

Precise definitions and many other consistencies can be found in the wonderful but sadly unmaintained Global Constraint Catalog[BCR05].

Values that do not respect these consistencies can safely be removed from the domain, thus reducing the search space, as they cannot be part of a solution to the problem.

Specialized algorithms are constructed for each type of constraints in order to enforce these consistencies; they are generally polynomial-time algorithms. A large portion of the CP literature is devoted to these algorithms.

The choice of consistency generally depends on the complexity of the algorithm that can enforce the consistency. While an AC consistency reduces the domains more than a bound consistency, the first might need an exponential algorithm while the second uses a polynomial one. An important part of CP modelization is to choose the right propagator and to balance computation time and pruning.

1.2.2 Constraint propagation

A CSP/COP contains in general multiple constraints, and in a CP context each of them is linked to one or multiple consistencies (with algorithms enforcing them). Calling one of these algorithm may do nothing on the domains, but calling another may remove a value that would allow the first algorithm to prune another domain.

This process then calls for a fixed-point: running all algorithms enforcing consistencies again and again, until no changes are made to the domain, reaching the point where no new information can be gathered by the consistencies used. Generally, the problem has still a large number of values present in its domain at the end of a fixed-point, thus the need for search algorithms (see next subsection).

This very simple fixed-point works, but in practice is slow as it runs multiple times algorithms whose variables in their scope have not seen changes in their domain.

This observation is the basis of constraint propagation[RVW06]. Multiple fixed-point algorithms exist and use this trick, such as AC-3[Mac81], AC-5[HDT92], and many others. They all focus on calling consistency algorithm only if they have a chance of effectively further prune domains, i.e. only when a domain from a variable in their scope has been modified.

1.2.3 Search and backtracking

Sadly, most of the time constraint propagation is not sufficient to empty a domain (proving the problem unsatisfiable) or to leave only a single value in each domain (finding a unique solution). It is thus necessary to explore the search space. CP solvers generally use complete methods for this, and rely on *backtracking*, i.e. they are able to revert previous decision. The search space is explored via the use of a search tree, where a node contains the current state of the domains (along with some additional constraints in some solvers). Once a fixed-point is reached, a decision is taken (*a branch is created*); this decision actually removes a part of the search space, typically by assigning a variable to a specific value in its domain. This part of the search space is then explored recursively; once this is done, the algorithm *backtracks* (rollbacks the changes made to the domains when the decision was taken) so that the solver state is again the same as before the decision. Then it takes the opposite decision (typically adding the constraint that the variable cannot take the value selected before) and recursively explores the new node.

The backtracking can either be implemented using techniques called *copying* or *trailing*:

- Copying simply makes a copy of the state of the solver before a decision is taken, this (full) copy being restored when a backtrack occurs.

- Trailing can be summarized as a copy-on-write mechanism: every variable or state in the solver additionally stores the last time it was modified. This time is actually a monotonically-increasing counter that is increased each time a decision or a backtrack is done. When a variable is modified, it checks whether the last time it was modified is equal to the current counter; if not, it stores its current value in a structure called the *trail*. The trail stores the modification made at each node separately (only for node on the path from the current node to the root); the modification can thus be reverted by simply visiting the last trail entries.

Implementing trailing is seen as more complex and requires all components having state in the solver to be aware of its usage, but uses far less memory and computation time in general. Copying is slower and more memory hungry, but has many advantages in the context of parallel solvers. See [Sch99] for an in-depth discussion.

The choice of the decision to be taken each node is made using heuristics, called a search heuristic or a branching heuristic. There exists multiple generic heuristics, from the simplest (binary static heuristic, taking a static order of variable, always branching on the first *unbound* (with more than one value in its domain) variable), to the more complex (see for example conflict ordering search[Gay+15]), through simple middle-grounds like the first-fail[HE80] principle. There is a large part of the CP literature focusing on these generic heuristics. Numerous well-known problems also have ad-hoc branching heuristics.

We redirect to the Handbook of Constraint Programming [RVW06] for a (far) more complete description of the basic and more advanced techniques and ideas used in CP.

1.3 Linear Programming

Linear programming (LP) focuses on optimizing linear problems of the following form[Mur]:

$$\max_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{1.1}$$

$$\text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{1.2}$$

$$\boldsymbol{x} \geq 0. \tag{1.3}$$

with $\boldsymbol{x}, \boldsymbol{b}, \boldsymbol{c}$ being vectors of reals and $\boldsymbol{A}$ a matrix of reals. The (linear) function $\boldsymbol{c}^T \boldsymbol{x}$ is called the *objective function*, while the inequalities formed by $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ are called the (linear) *constraints*.

Each constraint (i.e. each row of the matrix $\boldsymbol{A}$ along with its corresponding entry in b) can be viewed as an hyperplane cutting the space of all possible vectors $\boldsymbol{x}$, forming a *convex polyhedron* containing all the *admissible* (respecting the constraints) solution. The goal is thus to find a point (an admissible vector $\boldsymbol{x}$) inside this polyhedron that maximizes the objective function. Finding such a point is not always feasible: if the resulting polyhedron is empty (if the constraints are inconsistent), or if the polyhedron is unbounded in the direction of the optimization (it may happen that always increasing a variable increases the objective function, and that there is no upper limit for this variable).

An interesting property of linear problems is that any locally maximum solution is also a global maximum.

LP is a very active research area, and many algorithms and variants of them exists to solve LP problems. The most well known include the Simplex algorithm, which follows the edges of the convex polyhedron to find a minimum, and the interior point method-based algorithms, which move through the interior of the polyhedron. LP problems are in P, meaning they can be solved in polynomial time. In practice, problems with several thousand variables and constraints can sometimes be solved to optimality.

1.4 (Mixed-)Integer Linear Programming

An extension of linear programming is known as the Mixed-Integer (Linear) Programming (MIP or sometimes MILP), which adds *integral constraints*:

$$\max_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{1.4}$$

$$\text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{1.5}$$

$$\boldsymbol{x} \geq 0 \tag{1.6}$$

$$x_i \in \mathbb{Z}\ \forall i \in I \tag{1.7}$$

with I a subset of the indices of $\boldsymbol{x}$. If I contains all the indices, the problem is said to be an Integer Linear Programming (ILP) problem. Solving MIP/ILP problems is NP-Hard[Wol20c] (most NP-Hard problems have a direct encoding in MIP).

MIP solvers (such as Gurobi [Gur18] used extensively in this thesis) traditionally rely on *relaxing* the MIP problem by removing the integral constraints, thus obtaining a LP problem; solving this relaxed problem thus gives an upper bound to the original problem.

Thet then use a branch-and-cut[Wol20d] algorithm composed of

- a branch-and-bound[Wol20a], based on the upper bound described above,
- the cutting planes method[Wol20d] which add new linear constraints that remove parts of the linear polytope that are not into its integral counterpart.

For more information about the (numerous) properties and methods to solve LP and MIP problems, see [Wol20e], which offers a very pedagogic overview.

1.5 Overview of relaxations techniques used

A substantial part of this thesis is focused on finding *lower and upper bounds* (i.e. under- or overestimations that actually do not under- / over-estimate too much) to the optimal results of maximization problems, in order to avoid exploring part of the search space that don't contain better solutions than the one we currently have, and/or to generate suboptimal yet good solutions to these problems.

A common technique is to *relax* the problem being tackled, by removing large parts of the search space (producing lower bounds) or hard-to-satisfy constraints (producing upper bounds). In this thesis, an extensive use of this kind of techniques is made. Among them, two more complex techniques are column generation and Lagrangian relaxation.

1.5.1 Column generation

Let us take as an example the following MIP problem:

$$Z = \max_{x_1,\ldots,x_K} \sum_{i=1}^{K} c_i x_i \tag{1.8}$$

$$\text{subject to} \sum_{i=1}^{K} a_i x_i \leq b \tag{1.9}$$

$$x_i \leq d_i \qquad \forall i \in 1..K \tag{1.10}$$

$$x_i \geq 0 \qquad \forall i \in 1..K \tag{1.11}$$

$$\ldots \tag{1.12}$$

with possibly additional constraints acting on each variables individually, and a very large number of variables (K can be exponentially large depending on the problem being solved).

This kind of problem has a potentially intractable number of variables, linked by a very small number of constraints (here only equation (1.9)) and the objective values, with additional constraints only acting on one variable at a time.

Column generation[Wol20b] is a technique to solve such problems, by reducing the large set of variables to a tractable one, and by generating potentially useful (that may contribute to the objective) variables on the fly. The initial problem is called the Master Problem, and the new one with a limited number of variables is called the Restricted Master Problem (RMP).

In order to find which variable to insert next in the RMP, a common technique is to linearize the RMP (by removing integrality constraints), to solve the LP problem resulting from this relaxation, and to use the *dual* of the problem to create a new variable that has a positive *reduced cost*[2]. An example of this process will be presented in chapter 5.

1.5.2 Lagrangian relaxation

Some constraints are sometimes difficult to satisfy; a generic way to deal with that is to relax those and put them in the objective value, *penalizing* it when the constraint is not satisfied.

Given a MIP problem with the following form:

$$Z = \max_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{1.13}$$
$$\text{subject to } \boldsymbol{Ax} \leq \boldsymbol{b} \tag{1.14}$$
$$\boldsymbol{Cx} \leq \boldsymbol{d} \tag{1.15}$$
$$\boldsymbol{x} \geq 0 \tag{1.16}$$
$$x_i \in \mathbb{Z}\ \forall i \in I \tag{1.17}$$

with Z the optimal value. We can relax the constraint $\boldsymbol{Cx} \leq \boldsymbol{d}$ using the Lagrangian multipliers[Wol20f] $\boldsymbol{\lambda} \geq 0$:

$$Z(\boldsymbol{\lambda}) = \max_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} + \boldsymbol{\lambda}^T(\boldsymbol{d} - \boldsymbol{Cx}) \tag{1.18}$$
$$\text{subject to } \boldsymbol{Ax} \leq \boldsymbol{b} \tag{1.19}$$
$$\boldsymbol{x} \geq 0 \tag{1.20}$$
$$x_i \in \mathbb{Z}\ \forall i \in I \tag{1.21}$$

For any vector $\boldsymbol{\lambda}$ with positive values, we have that such a formulation is a (Lagrangian) *relaxation* of the initial problem: we have (for this maximization problem) that $Z(\boldsymbol{\lambda}) \geq Z\ \forall \boldsymbol{\lambda} \geq 0$. Of course one generally wants to find the best bound possible, by solving what's called a *Lagrangian dual problem*:

$$\min_{\boldsymbol{\lambda}} Z(\boldsymbol{\lambda}) \tag{1.22}$$

A classical way to find the best vector $\boldsymbol{\lambda}$ (or at least a very good one) is to use a subgradient method. [Wol20f] provides an in-depth discussion of the technique and of the theory around it.

[2]We don't go into details here, but duals of linear problems are also linear problems where each variable of the *primal* is represented by a constraint in the dual and conversely. The dual of the dual is the primal problem. An interesting property is that both problem are feasible, they share their optimal value. Duality theory is central in solving LP problems and in the field of optimization in general, see [Wol20f] for an in-depth, pedagogic review.

Part I

Mining a single submatrix

The Maximum-Sum Submatrix problem 2

This chapter is largely based on the paper G. Derval and P. Schaus. "Maximal-Sum Submatrix search using a hybrid Constraint Programming/Linear Programming approach". In: *European Journal of Operational Research* (2021). issn: 0377-2217. doi: `https://doi.org/10.1016/j.ejor.2021.06.008`. url: `https://www.sciencedirect.com/science/article/pii/S0377221721005142`..

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 [Med+02], the gene expressions for a set of patients [Van+02], the bilateral migration between two sets of countries [The18; Dao+18b], ranked tiling [Le +14], 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 [BSD17b] to answer this question. A 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). 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, Schaus, and Dupont [BSD17b], the MSS problem shares similarities with the biclustering one [Har72b; MO04] 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 [BSD17b], as it encodes the maximum edge-weighted biclique problem [Pee03].

The state-of-the-art results [BSD17b] for MSS are obtained using an advanced Constraint Programming (CP) approach that combines a custom filtering algorithm with a Large Neighborhood Search (LNS) [Sha98]. 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, Lodi, and Milano [FLM99]. Second, we demonstrate the use of Lagrangian relaxation for this problem, by finding another set of tighter upper bounds, 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.

Chapter's contributions

The contributions made in this chapter are a constraint programming framework for solving the Maximum-sum submatrix problem, including:

- three upper bounds of increasing tightness, along with results linking them, and efficient algorithms to compute them in the context of partial solutions;
- algorithms using these bounds to prune the search space, via dominances;
- a technique to reduce the size of the problem in the context of partial solutions;
- a large neighborhood search algorithms to find solutions on large instances.

2.1 Definitions and notations

Sets and multisets are both represented as uppercase characters (S). Vectors 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 (and zero). The set of rows and columns of the matrix are defined as $L_R := \{1, \dots, m\}$, $L_C := \{1, \dots, n\}$, 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. i is always an index of a row, and j is always an index of a column.

Definition 4. The Maximal-Sum Submatrix Problem. *A Maximal-Sum Submatrix (MSS) is a submatrix $\boldsymbol{M}_{R^*,C^*}$, with $R^* \subseteq L_R$ and $C^* \subseteq L_C$, such that:*

$$(R^*, C^*) = \underset{I \subseteq L_R, J \subseteq L_C}{\operatorname{argmax}} \sum_{i \in I, j \in J} M_{i,j} \tag{2.1}$$

Example 1. *Given the following matrix:*

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

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 [BSD17b] 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*

$$R^* = \{i \in L_R \mid \sum_{j \in C} M_{ij} > 0\} \tag{2.2}$$

2.2 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.

2.2.1 Maximum Weighted Edge Biclique (MWEB)

The input matrix of the MSS can be seen as an adjacency matrix of a bipartite weighted graph with possibly negative weights (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 5. Maximum Weighted Edge Biclique (MWEB) problem *[Tan08] 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 [Tan08] 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 [1]). This result thus also holds for MSS. The unweighted version of the problem, the Maximum Edge Biclique (MEB), is NP-Hard [Pee03]. It has been used to describe many variants of the biclustering problem (see Tanay, Sharan, and Shamir [TSS02] for an example, or the survey from Madeira and Oliveira [MO04]).

2.2.2 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 and Oliveira [MO04], 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 [PGA15]. Another survey by Xie et al. [Xie+18] focuses on the applicability of the biclustering algorithms in biological and biomedical data.

[1] RP, Randomized Polynomial-Time, is the set of decision problems solvable by a probabilistic Turing machine in polynomial time for which, if the correct answer is no, then always answers no, and if the answer is yes, the probability for answering yes is at least 1/2.

Some pattern mining algorithms can also be viewed as biclustering binary matrices. Frequent Itemset Mining[Agr+96; Han+04; SAG17] 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.

Tiling [GGM04] 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 (number of rows). Le Van et al. [Le +14] 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.

2.3 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 $\boldsymbol{M}$:

$$\sum_{i,j} \max(M_{ij}, 0). \tag{2.3}$$

It was used by Branders, Schaus, and Dupont [BSD17b] 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[GNS09]. We then prove that this bound is tighter than the upper bound presented by Branders, Schaus, and Dupont [BSD17b].

Branders, Schaus, and Dupont [BSD17b] 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

$$p_i = \begin{cases} \sum_j M_{ij} \cdot c_j & \text{if } r_i = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

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 $\mathsf{F}_{\mathsf{BigM}}$ hereafter) is given next.

$$\begin{aligned} & \max \textstyle\sum_i p_i && && \text{(2.5a)} \\ p_i \leq & & r_i \cdot up_i & \quad \forall i && \text{(2.5b)} \\ p_i \leq & (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i & & \quad \forall i && \text{(2.5c)} \\ r_i, c_j \in & & \{0, 1\} & \quad \forall i, j && \text{(2.5d)} \end{aligned}$$

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 (2.5b). The right part of constraint (2.5c) becomes $(\sum_j M_{ij} \cdot c_j) + lo_i$ which is greater than 0 by construction, and thus less constraining than (2.5b). If $r_i = 1$, then an upper bound for the contribution p_i is up_i (thus, in this case, (2.5b) is not constraining p_i). Constraint (2.5c) 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 $\mathsf{F}_{\mathsf{BigM}}$ (2.5), 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, $\mathsf{F}_{\mathsf{BigM\text{-}linear\text{-}rows}}$. It has interesting properties:

Theorem 2.3.1. *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*

$$r_i^* := \frac{lo_i + \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{2.6}$$

Proof. From (2.5b) and (2.5c), and as we must maximize p_i

$$p_i = \min(r_i \cdot up_i, (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i). \tag{2.7}$$

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 concave function (the minimum of two linear functions is a concave function), and its maximum is reached when both components are equal:

$$r_i \cdot up_i = (\textstyle\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i \tag{2.8}$$
$$r_i = \frac{lo_i + \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{2.9}$$

□

Moreover, the optimal contribution is thus

$$p_i^* = up_i \cdot r_i^* = \frac{up_i \cdot lo_i + up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{2.10}$$

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

Theorem 2.3.2. *$F_{BigM\text{-}linear\text{-}rows}$ has an optimal objective of*

$$\sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_j \max\left(0, \sum_i \frac{up_i \cdot M_{ij}}{up_i + lo_i}\right). \tag{2.11}$$

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 (2.5b) and (2.5c), 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 2.3.1). The objective becomes

$$\sum_i p_i = \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_i \frac{up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i} \tag{2.12}$$
$$= \sum_i \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_j c_j \cdot (\sum_i \frac{up_i \cdot M_{ij}}{up_i + lo_i}) \tag{2.13}$$

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 2.3.3. *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 (2.12).

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

□

It is also non-symmetric, hence 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* $F_{\textit{BigM-linear-rows}}$ *is* 6*, while it is* 7 *with* $\boldsymbol{M}^T$*. The optimum is* 6*.*

2.4 A Lagrangian-based upper bounding procedure

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

$$\max \sum_{i \in L_R, j \in L_C} M_{ij} \cdot x_{ij} \tag{2.14a}$$

$$x_{ij} \leq r_i \quad \forall i,j \tag{2.14b}$$

$$x_{ij} \leq c_j \quad \forall i,j \tag{2.14c}$$

$$r_i + c_j \leq x_{ij} + 1 \quad \forall i,j \tag{2.14d}$$

$$r_i, c_j, x_{ij} \in \{0,1\} \quad \forall i,j \tag{2.14e}$$

Rows and column selection are represented by variables r_i and c_j. x_{ij} indicates if the cell i, j is selected. Constraints (2.14b) and (2.14c) ensure that if a cell is selected, then the associated row and column are selected. Constraint (2.14d) 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 (2.14b) or (2.14c) will be trivial tight for this particular (i, j);

- Or $M_{ij} < 0$ and the value x_{ij} will be minimized, thus tightening constraint (2.14d).

Constraints (2.14b), (2.14c) and (2.14d) can thus be rephrased without loss of generality as

$$\begin{aligned} x_{ij} \leq& \quad r_i & \forall i,j : M_{ij} > 0 \qquad (2.15a)\\ x_{ij} \leq& \quad c_j & \forall i,j : M_{ij} > 0 \qquad (2.15b)\\ r_i + c_j \leq& \, x_{ij} + 1 & \forall i,j : M_{ij} < 0. \qquad (2.15c) \end{aligned}$$

The linear relaxation of F_{x} (named hereafter $\mathsf{F}_{\text{x-linear}}$) uses more variables than $\mathsf{F}_{\text{BigM-linear-rows}}$ ($\mathcal{O}(|L_R| \cdot |L_C|)$ 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, as it is shown in the experiments below.

It is however possible to use a Lagrangian relaxation of 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 (2.15a), (2.15b) and (2.15c). 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, $\mathsf{F}_{\text{x-lrelax-all}}$:

$$\begin{aligned} \min_{\alpha_{ij},\beta_{ij},\gamma_{ij}} \max_{r_i,c_j,x_{ij}} \sum_{ij} M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) + \beta_{ij}(c_j - x_{ij}) \\ + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) \qquad (2.16a) \end{aligned}$$

$$\begin{aligned} \text{with} \quad r_i, c_j, x_{ij} \in \{0,1\} & \quad \forall i,j & (2.16b)\\ \alpha_{ij}, \beta_{ij}, \gamma_{ij} \geq 0 & \quad \forall i,j & (2.16c)\\ \alpha_{ij} = \beta_{ij} = 0 & \quad \forall i,j : M_{ij} \leq 0 & (2.16d)\\ \gamma_{ij} = 0 & \quad \forall i,j : M_{ij} \geq 0 & (2.16e) \end{aligned}$$

The constraint (2.16b) 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. This is allowed by the fact the variables r_i, c_j and x_{ij} are only constrained by this constraint; the maximization process on the relaxed problem will always push them to either 0 or 1 as a consequence, depending on their coefficient in the objective function.

By the Strong Lagrangian Duality property [BBV04] (that implies that Lagragian relaxations of convex problems respecting Slater conditions, which holds in the *linear* problem $\mathsf{F}_{\text{x-linear}}$, have the same objective value as the original problem), its optimum is thus the same as $\mathsf{F}_{\text{x-linear}}$.

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 resolution time in practice.

We show below another Lagrangian relaxation, $\mathsf{F}_{\text{x-lrelax-partial}}$, which does not relax all the constraints and as consequence uses fewer 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.

$$\begin{aligned}
\min_{\alpha_{ij},\gamma_{ij}} \max_{r_i,c_j,x_{ij}} \sum_{ij} & M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) \\
& + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) && \text{(2.17a)} \\
\text{with} \qquad x_{ij} &\le c_j \qquad \forall i,j && \text{(2.17b)} \\
r_i, c_j, x_{ij} &\in \{0,1\} \quad \forall i,j && \text{(2.17c)} \\
\alpha_{ij}, \gamma_{ij} &\ge 0 \qquad \forall i,j && \text{(2.17d)} \\
\alpha_{ij} &= 0 \qquad \forall i,j : M_{ij} \le 0 && \text{(2.17e)} \\
\gamma_{ij} &= 0 \qquad \forall i,j : M_{ij} \ge 0 && \text{(2.17f)}
\end{aligned}$$

Again it can be shown that the constraint (2.17c) can be relaxed into a linear version while not modifying the optimum solutions (by the same argument as before), and is then equivalent to $\mathsf{F}_{\text{x-linear}}$. The same sub-gradient method can be used as the Lagrangian subproblem is still convex on the Lagrangian multipliers. However, for a given set of Lagrangian multipliers, the Lagrangian subproblem cannot be solved as trivially as before. Let us give a name to the Lagrangian subproblem, and rearrange its formulation:

$$f(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \max_{r_i,c_j,x_{ij}} \sum_{i,j} M_{ij} \cdot x_{ij} + \alpha_{ij}(r_i - x_{ij}) + \gamma_{ij}(x_{ij} + 1 - r_i - c_j) \tag{2.18}$$

$$\begin{aligned}
&= \max_{r_i,c_j,x_{ij}} \sum_{i,j} \left(x_{ij} \cdot (M_{ij} - \alpha_{ij} + \gamma_{ij}) + \gamma_{ij} \right) \\
&\qquad + \sum_i r_i \cdot (\sum_j \alpha_{ij} - \gamma_{ij}) - \sum_j c_j \cdot (\sum_i \gamma_{ij}) \qquad \text{(2.19)}
\end{aligned}$$

such that $x_{ij} \le c_j \; \forall i,j$

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 (2.20) 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

$$\sum_i \max(0, M_{ij} - \alpha_{ij} + \gamma_{ij}) - \gamma_{ij}. \tag{2.20}$$

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 dual function, by directly applying the resulting formula:

$$\begin{aligned} f(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = &\sum_i \max(0, \sum_j \alpha_{ij} - \gamma_{ij}) + \\ &\sum_j \max(0, \sum_i \max(0, M_{ij} - \alpha_{ij} + \gamma_{ij}) - \gamma_{ij}) + \sum_{i,j} \gamma_{ij} \end{aligned} \tag{2.21}$$

Overall, the subproblem of $\mathsf{F}_{\text{x-lrelax-partial}}$ can be solved in the same complexity as $\mathsf{F}_{\text{x-lrelax-all}}$ (i.e. $\mathcal{O}(|L_R| \cdot |L_C|)$) but with fewer Lagrangian multipliers, ensuring a faster solving time for the subgradient algorithm, while preserving the exact same optimal objective value. The algorithm to minimize $\mathsf{F}_{\text{x-lrelax-partial}}$ is given in Algorithm 3.

The algorithm uses a simple subgradient-descent algorithm with a harmonic update [CF15] of step size μ. By default we use a multiplicative factor of 0.95 for the update, and start with $\mu_0 = 1$. The optimality gap between the optimum of $\mathsf{F}_{\text{x-linear}}$ and the one computed by the subgradient algorithm on random matrices filled with random Gaussian noise is shown in Figure 2.1. We experimentally observe on that Figure that progress heavily slows after 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

Algorithm 3 Solving $F_{\text{x-lrelax-partial}}$

```
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 $i \in L_R$, $j \in L_C$ do                      ▷ Base cell contribution
        ub $\leftarrow$ ub $+ \gamma_{ij}$

    for all $i \in L_R$ do                                   ▷ Select all rows with positive contribution
        contribution $\leftarrow \sum_j \alpha_{ij} - \gamma_{ij}$
        if contribution $> 0$ then
            $r_i \leftarrow 1$
            ub $\leftarrow$ ub + contribution

    for all $j \in L_C$ do                                   ▷ Select all columns with positive contribution
        contribution $\leftarrow 0$
        for all $i \in L_R$ 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 $> 0$ then
                contribution $\leftarrow$ contribution + cellContribution
                $x_{ij} \leftarrow 1$

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

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

function subgradientDescent(nIterations)
    $\boldsymbol{\alpha}, \boldsymbol{\gamma} \leftarrow$ random initialization (uniform between 0 and 1)
    $\mu \leftarrow 1$
    for nIterations iterations do
        ub, $\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{x} \leftarrow$ solveSubproblem($\boldsymbol{\alpha}, \boldsymbol{\gamma}$)   ▷ Note that here, at any point of the algo-
        $\mu \leftarrow 0.95 \cdot \mu$                                                                                          rithm, ub is a valid upper bound
        for all $i \in L_R$ do
            for all $j \in L_C$ do
                $\alpha_{ij} \leftarrow \max(0, \alpha_{ij} - \mu \cdot (r_i - x_{ij}))$
                $\gamma_{ij} \leftarrow \max(0, \gamma_{ij} - \mu \cdot (x_{ij} + 1 - r_i - c_j))$
    return solveSubproblem($\boldsymbol{\alpha}, \boldsymbol{\gamma}$)
```

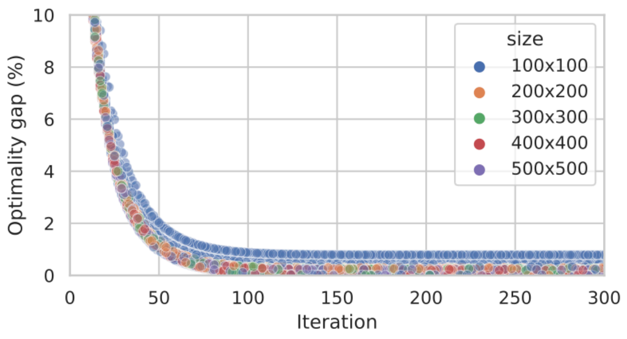


Figure 2.1: Optimality gap between $\mathrm{F}_{\text{x-linear}}$ and the value computed by the subgradient on 50 random matrices.

Table 2.1: Time (s) to reach a given optimality gap, for a black-box LP solver (Gurobi) using $\mathrm{F}_{\text{x-linear}}$ and a Python implementation of Algorithm 3 (subgradient on $\mathrm{F}_{\text{x-lrelax-partial}}$).

	LP			Lagrangian	
Size	Opt	5%	1%	5%	1%
100x100	42.54	0.18	0.18	0.77	1.77
200x200	908.7	1.17	1.17	2.73	6.09
300x300	271.9	209.4	252.3	6.85	14.9
400x400	1039	812.3	962.7	12.3	28.2
500x500	2391	1983	2277	15.4	34.7

experiment section). Existing research shows that even poor approximations or non-updated approximation of the Lagrangian multipliers may be beneficial [Sel04].

Table 2.1 shows the time taken to reach a given optimality gap, for a LP solver solving the problem with $\mathrm{F}_{\text{x-linear}}$ and Algorithm 3. There is an important speedup, which is expected as a lot of variables are removed in $\mathrm{F}_{\text{x-lrelax-partial}}$ w.r.t $\mathrm{F}_{\text{x-linear}}$.

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.

2.5 A note about bounds' tightness and relations

So far we discussed three different bounding procedure:

- The natural upper bound $\sum_{i,j} \max(M_{ij}, 0)$;

- The $F_{\text{BigM-linear-rows}}$ model, presented in section 2.3, 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 $F_{\text{x-linear}}$ model presented in section 2.4, 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 tightness. We showed in theorem 2.3.3 that $F_{\text{BigM-linear-rows}}$ is a tighter bound than the natural one. The following theorem shows that $F_{\text{x-linear}}$ is tighter than $F_{\text{BigM-linear-rows}}$.

Theorem 2.5.1. *The optimal objective of* $F_{\text{x-linear}}$ *is less or equal than the optimal objective of* $F_{\text{BigM-linear-rows}}$.

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

The optimal variables x_{ij} for $F_{\text{x-linear}}$ can be inferred from the row and column variables. From equations (2.14b), (2.14c) and (2.14d) we have the following constraints on x_{ij}:

$$\max(0, r_i + c_j - 1) \leq x_{ij} \leq \min(r_i, c_j) \tag{2.22}$$

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 $F_{\text{x-linear}}$ is then

$$\text{obj}_{\text{x-linear}} = \sum_i \Big(\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \Big) \tag{2.23}$$

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

From equations (2.5a), (2.5b) and (2.5c), we have that the solution is

$$\text{obj}_{\text{BigM-linear}} = \sum_i \min(r_i \cdot up_i, (\sum_j M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i) \tag{2.24}$$

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

$$\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \tag{2.25}$$

$$\leq \sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} \tag{2.26}$$

$$\leq \sum_{j|M_{ij}>0} r_i \cdot M_{ij} = r_i \cdot up_i \tag{2.27}$$

Moreover,

$$\sum_{j|M_{ij}>0} \min(r_i, c_j) \cdot M_{ij} + \sum_{j|M_{ij}<0} \max(0, r_i + c_j - 1) \cdot M_{ij} \tag{2.28}$$

$$\leq \sum_{j|M_{ij}>0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i + c_j - 1) \cdot M_{ij} \tag{2.29}$$

$$\leq \sum_{j|M_{ij}>0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i - 1) \cdot M_{ij} \tag{2.30}$$

$$\leq \sum_j c_j \cdot M_{ij} + \sum_{j|M_{ij}<0} (r_i - 1) \cdot M_{ij} = \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i \tag{2.31}$$

All of this for any row i. We thus have that for each row, the contribution of the row in $\text{obj}_{\text{x-linear}}$ is less or equal than in $\text{obj}_{\text{BigM-linear}}$. From equations (2.27) and (2.31):

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

$$\leq \min(r_i \cdot up_i, \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i) \ \forall i \tag{2.32}$$

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

$$\leq \sum_i \min(r_i \cdot up_i, \sum_j c_j \cdot M_{ij} + (1 - r_i) lo_i) \tag{2.33}$$

$$\implies \text{obj}_{\text{x-linear}} \leq \text{obj}_{\text{BigM-linear}} \tag{2.34}$$

□

It is thus expected that bounding and filtering based on $F_{\text{x-linear}}$ will prune the search space more than ones based on $F_{\text{BigM-linear}}$. Figure 2.2 gives a visual representation of all the bounds used in this chapter.

It is possible to show that all these linear bounds can be arbitrarily distant from the discrete problem's optimal solution, as shown in Example 3.

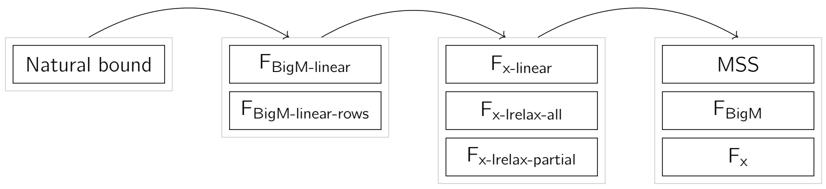


Figure 2.2: Summary of all the models used in this chapter. A gray rectangle means that enclosed methods are equivalent (have the same optimum objective). An arrow from A to B indicates that B has a lower optimum than A. This relation is transitive.

Table 2.2: Bounds obtained for example 3

Method	Value for (n, a, b) $(20, 1, 1000)$	$(20, 19, 1)$
Natural bound	20	380
$\mathrm{F}_{\text{BigM-linear}}$	$\simeq 19.9989$	190
$\mathrm{F}_{\text{x-linear}}$	10	190
Optimum MSS	1	100

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 if $a \leq b$, and $ca - (c^2 - c)b$ with c the best integer around $\frac{a+b}{2b}$ otherwise. Table 2.2 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 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 $\mathrm{F}_{\text{x-linear}}$, and produce the aforementioned objective value. This can be at most $\frac{\min(m,n)}{2}$ times greater than the actual discrete optimum.

2.6 Using the bounds in a CP framework

We reuse the framework introduced by [BSD17b]. It is based on Constraint Programming which, in this particular case, is based on a Depth First Search using Branch-and-Bound on the space of the possible assignment of the variables (rows and columns to select).

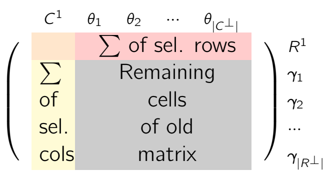
We use one binary variable per row (r_1, ..., r_m) and per column (c_1, ..., c_n). These binary variables can either be assigned ($v = 1$), unassigned ($v = 0$) or undecided yet ($v = \perp$). At each DFS iteration, an undecided variable is selected (according to a *search strategy*, described in the 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.

2.6.1 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:

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



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 [Sha98], most of the variables are decided during the search.

2.6.2 Update of the incumbent solution

As observed in [BSD17b], 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

$$\sum_{i \in R^1} \sum_{j \in C^1} M_{ij} + \sum_{i \in R^\perp} \max(0, \sum_{j \in C^1} M_{ij}) \tag{2.35}$$

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[2].

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 $\mathcal{O}(n+m)$.

2.6.3 Upper bounding

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

2.6.3.1 $\mathbf{F_{\text{BigM-linear-rows}}}$

Equation (2.11), 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:

$$\sum_{i \in R^\perp} \frac{up_i \cdot lo_i}{up_i + lo_i} + \sum_{j \in C^1} \Big(\sum_{i \in R^1} M_{ij} + \sum_{i \in R^\perp} \frac{up_i \cdot M_{ij}}{up_i + lo_i} \Big) + \sum_{j \in C^\perp} \max(0, \sum_{i \in R^1} M_{ij} + \sum_{i \in R^\perp} \frac{up_i \cdot M_{ij}}{up_i + lo_i}) \tag{2.36}$$

[2]Note that [BSD17b] 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 4 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 ($\perp \to 0/1$)
    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 ($\perp \to 0/1$)
    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
        $\texttt{solC} \leftarrow \texttt{solC} + \max(0, \texttt{colVal}[j])$
    $\texttt{incumbentSolution} \leftarrow \max(\texttt{incumbentSolution}, \texttt{solR}, \texttt{solC})$
```

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:

$$up_i := \sum_{j \in C^1} M_{ij} + \sum_{j \in C^\perp} \max(0, M_{ij}) \tag{2.37}$$

$$lo_i := -\sum_{j \in C^1} M_{ij} + \sum_{j \in C^\perp} \max(0, -M_{ij}) \tag{2.38}$$

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 "$\text{F}_{\text{BigM-linear-rows}}$ (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 5. The name of this upper bounding procedure is "$\text{F}_{\text{BigM-linear-rows}}$ (fixed)" in the experiments presented in the last section.

2.6.3.2 $\text{F}_{\text{x-lrelax-partial}}$

The model presented at equation (2.17) 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 3, shown in Algorithm 6. The algorithm runs in $\mathcal{O}(kmn)$ at each node of the tree, where k is the number of subgradient descent steps.

2.6.4 Reduced-Cost-based filtering

2.6.4.1 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 $ub^{R^1, R^\perp \setminus \{i\}, C^1, C^\perp}$).

Algorithm 5 Implementation of $F_{\text{BigM-linear-rows}}$ (fixed) with caching

```
                                                              ▷ Initialization
curBound ← 0          ▷ Reversible (i.e. reverted to previous value when a backtrack occurs)
curColBound[$j$] ← 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[$i$] ← $\frac{up_i \cdot lo_i}{up_i + lo_i}$
    for all $j \in L_C$ do
        cellContribution[$i, j$] ← $\frac{M_{ij} \cdot lo_i}{up_i + lo_i}$
        curColBound[$j$] ← curColBound[$j$] + cellContribution[$i, j$]
    curBound ← curBound + rowContribution[$i$]

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

function onColModified(i)    ▷ Called when a column is selected/unselected ($\perp \to 0/1$)
    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 ← 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 ← curBound − curColBound[$j$]

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

        if $j \in C^1$ then
            ▷ If the column is selected, then its contribution is updated.
            delta ← 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 ← 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 ← delta − oldValue
        else if oldValue > 0 $\wedge$ curColBound[$j$] > 0 then
            ▷ If the column was selected, and still is, update the contribution.
            delta ← delta + cdelta
    curBound ← curBound + delta
```

Algorithm 6 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
            for all $i \in L_R$ do          column, we must reset the cells to 0.
                $x_{ij} \leftarrow 0$

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

We then have these filtering rules:

$$ub^{R^1 \cup \{i\}, R^\perp \setminus \{i\}, C^1, C^\perp} < \text{best} \Rightarrow i \notin R \tag{2.39}$$

$$ub^{R^1, R^\perp \setminus \{i\}, C^1, C^\perp} < \text{best} \Rightarrow i \in R \tag{2.40}$$

where best is the value of the best solution found so far.

Explained differently, 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 [FLM99].

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 chapter 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 $\mathsf{F}_{\text{BigM-linear-rows}}$ (see Section 2.6.3.1). 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 $\mathsf{F}_{\text{BigM-linear-rows}}$: one with the row filtering, one without.

2.6.4.2 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 (2.37) and (2.38).

The following filtering rules always apply[3]:

$$up_i < 0 \Rightarrow i \notin R^* \tag{2.41}$$

$$lo_i < 0 \Rightarrow i \in R^* \tag{2.42}$$

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 (2.41) was introduced by [BSD17b], while (2.42) is a new contribution. This is straightforward, but was not possible in the implementation of [BSD17b] 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.

2.6.5 Methods and complexities summary

Table 2.3 provides a summary of all the methods and their complexities. Table 2.4 shows all the combinations of methods used in the experiments.

Table 2.3: Methods and complexities (computed at each node, non-symmetric)

		Upper	Filtering		
Model	Type	bounding	LB-Low	Row UB	Column UB
Base [BSD17b]		$\mathcal{O}(n\Delta_r + \Delta_c)$	$\mathcal{O}(\Delta_r + \Delta_c)$	$\mathcal{O}(mn\Delta)$	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 2.4: Combinations of methods used in the experiments

	Overall	Applied	Enabled filtering		
Name	complexity	models	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 2.6.4.2. *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[4] of the search tree ($s\%$). The chosen pairs (k, s) are:

- $(50, 100\%)$,
- $(100, 100\%)$,
- $(150, 100\%)$,
- $(150, 30\%)$,
- $(150, 60\%)$,
- $(150, 90\%)$,
- $(150, 95\%)$.

2.7 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 (2.5).

2.7.1 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 2.4), 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

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

[4]A full SGD is always run at the root node

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, since both are relevant.

Table 2.5 shows the ratio for all datasets and matrix sizes. As it 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 fewer 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.

Table 2.5: 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

2.7.2 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, [Sha98]) with an adaptive relaxation rate. The additional constraint (maximum running time and timeout per LNS iteration) and results are shown in Table 2.6 and Figure 2.3, 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 diversification method[5] 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 2.6. Figure 2.3 shows, however, that the convergence rate is higher for some methods. The group of methods that

[5]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

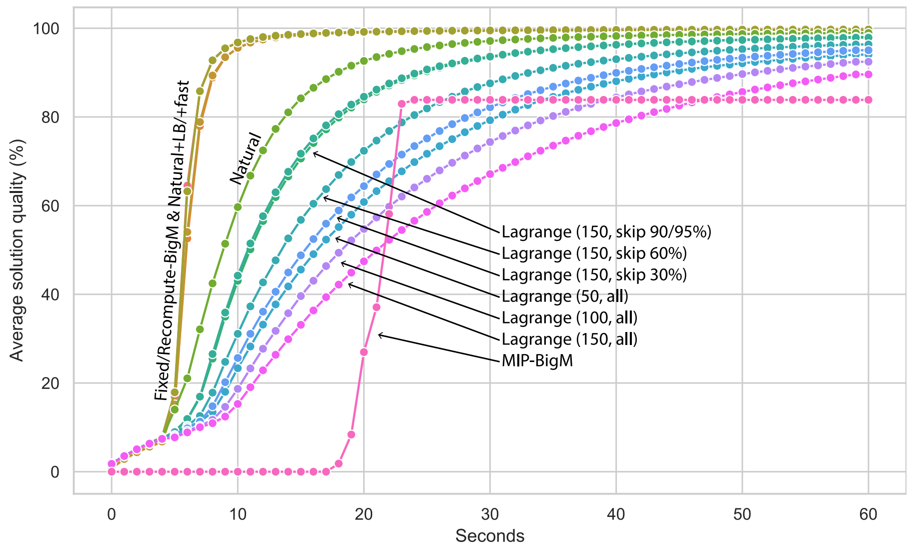


Figure 2.3: Average solution quality w.r.t. time for all methods. (MIP-Strong is omitted due to poor performance)

Table 2.6: 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%
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.

performs better can be summarized as follows:

- 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.

2.7.3 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

Table 2.7: 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

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 2.7, 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.

2.7.4 Real-life data

Table 2.8 shows runtime, reached objective value, and runtime at which the best objective value was reached, from multiple real-life datasets [Le +14; Bra+19a].

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 +14]), 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, [Bra+19a]) 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

Table 2.8: 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	4052763	(0.1s)	636s		✓
`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		✓

prove useful on this particular instance.

2.8 Chapter 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 tighter, 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[DS20a], along with the experiments code and results [DS20b]. The experiment were run on the OscaR CP solver [Osc12].

Cardinality constrained MSS problem | 3

At the moment this thesis is written, the work in this chapter is unpublished yet.

3.1 Constraining the choice of the submatrix

The MSS problem introduced in the previous chapter aims at finding the submatrix with the greatest sum, without particular constraints. In practice, users may want to constrain the size of submatrices or implement other constraints.

A principle used in [BSD17b] is to remove a constant value λ from all cells in the matrix; this idea came from the fact that the authors actually had matrices with positive entries. The MSS is thus not searched in the initial matrix $\boldsymbol{M}$ but in a matrix $\boldsymbol{M}'$ with $M'_{i,j} = M_{i,j} - \lambda$. This has two properties:

- All solutions with a positive objective value have a minimum average weight per cell of λ. This is easily proven by writing the objective function for a solution (R, C):

$$
\begin{aligned}
0 &\leq \sum_{(i,j)\in R\times C} M'_{i,j} \\
&= \sum_{(i,j)\in R\times C} (M_{i,j} - \lambda) \\
&= (\sum_{(i,j)\in R\times C} M_{i,j}) - |R \times C| * \lambda \\
\Rightarrow \lambda &\leq \frac{\sum_{(i,j)\in R\times C} M_{i,j}}{|R \times C|}
\end{aligned}
$$

- Submatrices with fewer cells (less area) are prioritized (as the λ penalty over the objective is proportional to the number of cells).

In practice, increasing λ thus reduces the area of the submatrices found, and choosing the lambda correctly to obtain a specific size can be cumbersome. Its interpretability is also somewhat difficult.

In this chapter we propose as an alternative to constraint directly the number of rows and columns of the searched submatrices (we call these the cardinality constraints). While this still imposes to the user to select two additional parameters instead of one, this allows a more fine-grained specification of the mining problem, especially instrumental when the extracted submatrix must be visualized in circular plots [Krz+09]. Indeed, over-cluttering becomes rapidly an issue with Chord Diagrams when there are too many connections displayed[Dao+21].

We thus introduce a variant of the MSS problem allowing specifying lower and upper bounds on the number of rows and columns to select in the submatrix:

Definition 6. The cardinality-constrained Maximum-Sum Submatrix Problem (C-MSS) *is the MSS problem with the additional constraints* $r_{min} \leq |R^*| \leq r_{max}$ *and* $c_{min} \leq |C^*| \leq c_{max}$.

Example 4. *Given the following matrix:*

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

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. With the additional cardinality constraints* $r_{\min} = c_{\min} = 0, r_{\max} = 3, c_{\max} = 2$*, the C-MSS is* $\boldsymbol{M}^{ex}_{\{1,2,4\},\{3,5\}}$ *(highlighted in grey), its value being 15.*

We discuss in this chapter an extension of the bounds developed for the MSS problem, for the C-MSS problem.

Chapter's contributions

The contributions made in this chapter are:

- a precise definition of the MSS with cardinality constraints;
- an upper bound for the problem based on a Big-M formulation of the problem, being partially solved by inspection;
- an implementation of this upper bound in a CP framework, including support for partial solutions
- an adaptation of heuristics presented in the last chapter
- experiments comparing our solver with an off-the-shelf MIP solver.

3.2 An upper bound for the C-MSS problem

The next theorem gives a simple upper bound for the C-MSS that is the counterpart of selecting all the positive entries in the standard MSS.

Theorem 3.2.1. *An upper bound for the C-MSS is given by*

$$\max_{\substack{R \subseteq L_R \\ r_{\min} \leq |R| \leq r_{\max}}} \sum_{i \in R} \max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} M_{ij} \tag{3.1}$$

Proof. $\max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} M_{ij}$ is an upper bound of the contribution of the row i, therefore by selecting the most favorable ones with the first max operator, we still have an upper bound. □

An alternative upper bound can be derived by solving a relaxation of the MIP problem presented in model (2.5), with the additional cardinality constraints:

$$\begin{aligned}
&& \max \textstyle\sum_i p_i && && \text{(3.2a)}\\
p_i \leq && r_i \cdot up_i && \forall i \in L_R && \text{(3.2b)}\\
p_i \leq && (\textstyle\sum_{j \in L_C} M_{ij} \cdot c_j) + (1 - r_i) \cdot lo_i && \forall i \in L_R && \text{(3.2c)}\\
\sum r_i \in && [r_{\min}, r_{\max}] && \forall i \in L_R && \text{(3.2d)}\\
\sum c_j \in && [c_{\min}, c_{\max}] && \forall j \in L_C && \text{(3.2e)}\\
r_i \in && \{0, 1\} && \forall i \in L_R && \text{(3.2f)}\\
c_j \in && \{0, 1\} && \forall j \in L_C && \text{(3.2g)}
\end{aligned}$$

with $up_i = \sum_{j \in L_C} \max(M_{ij}, 0)$ and $lo_i = -\sum_{j \in L_C} \min(M_{ij}, 0)$. In chapter 2, we found in Theorem 2.3.1 that the optimal value for each r_i and p_i in order to maximize the linear relaxation of this model (without cardinality constraints) are respectively

$$r_i^* = \frac{lo_i + \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{3.3}$$

$$p_i^* = up_i \cdot r_i^* = \frac{up_i \cdot lo_i + up_i \cdot \sum_j M_{ij} \cdot c_j}{up_i + lo_i}. \tag{3.4}$$

We first present a method to create a specific class of optimal solutions (i.e. a subset of the set of optimal solutions) to the LP relaxation of the problem. We then derive the upper bound. To begin, we need two small results, showing the behavior of the objective value w.r.t. small increases of a row contribution, in the *relaxed-rows* problem (where we relax equation (3.2f) in a linear way):

Theorem 3.2.2. *Increasing in isolation any $r_i < r_i^*$ by ϵ ($0 \leq \epsilon \leq r_i^* - r_i$) increases the contribution of the row, p_i, by $\epsilon \cdot up_i$. Decreasing in isolation any $r_i > r_i^*$ by ϵ ($0 \leq \epsilon \leq r_i - r_i^*$) increases the contribution of the row in the objective, p_i, by $\epsilon \cdot lo_i$.*

Proof. Direct from equations (3.2b), (3.2c) and (2.7), which is recalled below:

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

□

Corollary 3.2.2.1. *Any optimal solution of the relaxed-rows problem either respects $\forall i : r_i \leq r_i^*$ or $\forall i : r_i \geq r_i^*$.*

Proof. By contradiction. Let us assume we have two rows a, b such that $r_a < r_a^*$ and $r_b > r_b^*$. Let $\delta = \min(r_a^* - r_a, r_b - r_b^*)$. Increasing r_a by δ and decreasing at the same time r_b by δ (this simultaneous move ensures that the cardinality constraint is respected) would increase the objective by $\delta \cdot (up_a + lo_b) > 0$ while respecting all the constraints. □

Theorem 3.2.3. *Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be any ordering of the rows such that they are in decreasing order of upper bound, namely $up_{\alpha_a} \geq up_{\alpha_b}$ $\forall a \leq b$. Similarly, but for lower bounds, let $\beta_1, \beta_2, \ldots, \beta_m$ be any ordering such that $lo_{\beta_a} \leq lo_{\beta_b}$ $\forall a \leq b$. There exists an optimal solution to the relaxed rows MIP formulation (i.e. $r_i \in [0, 1]$ $\forall i$) such that at least one of these two properties holds:*

1. $\exists v \in L_R$ such that:

$$r_{\alpha_i} = \begin{cases} r^*_{\alpha_i} & \text{if } i < v \\ x & \text{if } i = v \\ 0 & \text{if } i > v \end{cases} \tag{3.5}$$

with $0 < x \leq r^*_{\alpha_v}$.

2. $\exists v \in L_R$ such that:

$$r_{\beta_i} = \begin{cases} 1 & \text{if } i < v \\ x & \text{if } i = v \\ r^*_{\beta_i} & \text{if } i > v \end{cases} \tag{3.6}$$

with $r^*_{\beta_v} \leq x < 1$.

Proof. By Corollary 3.2.2.1, we have that any optimal solution respects either $\forall i : r_i \leq r^*_i$ or $\forall i : r_i \geq r^*_i$.

Let us consider now that we are in the first case, $\forall i : r_i \leq r^*_i$. We prove that we can construct a solution respecting (3.5) (the proof is similar for the second case, this time respecting (3.6)). By construction, we might still have multiple rows such that $0 < r_i < r^*_i$. Note that for any such two rows a, b, we have that $A_a = A_b$ (as it would otherwise lead to a contradiction, the solution not being optimal).

We construct another optimal solution: select α_a, α_b in these rows such that $a < b$ and $r_{\alpha_a} < r^*_{\alpha_a}$, $r_{\alpha_b} < r^*_{\alpha_b}$. Increase r_{α_a} and decrease r_{α_b} by the same value $\epsilon = \min(r^*_{\alpha_a} - r_{\alpha_a}, r_{\alpha_b} - r^*_{\alpha_b})$. This does not change the objective value of the whole solution, respects all the constraints, and effectively enforces either $r_a = r^*_a$ or $r_b = r^*_b$. We can then repeat this process until it is not possible to find new candidates a and b.

At the end of this process, we have at most one row i which is $0 < r_i < r^*_i$. Moreover, the property (3.5) is respected, as if it was not, then there would exist rows α_a, α_b, $a < b$, such that $r_{\alpha_a} < r^*_{\alpha_a}$ and $r_{\alpha_b} = r^*_{\alpha_b}$. This is actually a contradiction, as increasing r_{α_a} and decreasing r_{α_b} would increase the objective value, as, by definition, $up_{\alpha_a} > up_{\alpha_b}$. □

We now prove that these solutions have strong relations with the row cardinality limits.

Theorem 3.2.4. *Exactly one claim below is true for any instance of the row-relaxed MIP formulation:*

(A) There exists an optimal solution such that $r_i = r^*_i \ \forall i$;

(B) *There exists an optimal solution such that it respects* (3.5)*, in which there exists a row i with $r_i < r_i^*$, and $\sum_i r_i = r_{\max}$;*

(C) *There exists an optimal solution such that it respects* (3.6)*, in which there exists a row i with $r_i > r_i^*$, and $\sum_i r_i = r_{\min}$.*

Proof. From Theorem 3.2.3, there exists an optimal solution such that it respects (3.5) or (3.6). If the solution respects both, it falls into claim (A). If this is not the case, then there exists a row i such that $r_i \neq r_i^*$.

Let us assume the solution respects only (3.5), and falls into claim (B). We now need to prove that this solution has $\sum_i r_i = r_{\max}$. This is easily done by contradiction: assume that $\sum_i r_i < r_{\max}$. Let $w = \alpha_v$ if $r_{\alpha_v} < r^*_{\alpha_v}$ or $w = \alpha_{v+1}$ otherwise. It is possible to increase r_w and remain feasible, increasing the objective value, hence the contradiction.

Claim (C) is proven in a similar fashion, using (3.6). $\square$

This last theorem can then be used to compute an upper bound for the problem. It is possible to generate three solutions, corresponding to each of the possibilities of the Theorem 3.2.4.

3.2.1 Case (A)

Case (A) is the case where the cardinality constraints $r_{\min}, r_{\max}$ are not tight, and can actually be removed. The objective value becomes:

$$\texttt{caseA}() := \sum_{i \in L_R} \tfrac{up_i \cdot lo_i}{up_i + lo_i} + \max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \sum_{i \in L_R} \tfrac{up_i \cdot M_{ij}}{up_i + lo_i} \tag{3.7}$$

This solution only exists if the sum of the r_i for all the rows respects the constraints:

$$r_{\min} \leq \sum_{i \in L_R} \tfrac{lo_i + \sum_{j \in C^*} M_{ij}}{up_i + lo_i} \leq r_{\max} \tag{3.8}$$

where C^* is the optimal selection of columns in equation (3.7).

3.2.2 Case (B)

Case (B) requires more attention. We have to find for which v the condition (3.5) is respected. For a given v, we can rewrite the objective according to the condition (3.5). Let us consider, without loss of generality, that the rows are ordered in decreasing order of up_i: $up_a \geq up_b \; \forall a \leq b$; this simplifies the notation w.r.t. the ordering defined in (3.5).

We thus have that $r_i = r_i^* \ \forall i < v$, $r_v < r_v^*$ and $r_i = 0 \ \forall i > v$. As we have $r_i \leq r_i^* \ \forall i$, it follows that $p_i = r_i \cdot up_i \ \forall i \leq v$. Moreover, we can infer the value of r_v from the other rows: $r_v = r_{\max} - \sum_{i<v} r_i^*$. We can then put everything together into the objective function:

$$(r_{\max} - \sum_{i<v} r_i^*) up_v + \sum_{i<v} r_i^* \cdot up_i \tag{3.9}$$

$$= r_{\max} \cdot up_v + \sum_{i<v} r_i^* \cdot (up_i - up_v) \tag{3.10}$$

$$= r_{\max} \cdot up_v + \sum_{i<v} \tfrac{lo_i \cdot (up_i - up_v)}{up_i + lo_i} + \sum_{j \in L_C} c_j \cdot (\sum_{i<v} \tfrac{(up_i - up_v) \cdot M_{ij}}{up_i + lo_i}) \tag{3.11}$$

We must maximize this objective function under the column cardinality and integer constraints ($c_{\min} \leq \sum_j c_j \leq c_{\max}$, $c_j \in \{0,1\}$) and the constraint on r_v:

$$0 \leq r_v = r_{\max} - \sum_{i<v} r_i^* \leq 1 \tag{3.12}$$

$$\iff r_{\max} - 1 \leq \sum_{i<v} r_i^* \leq r_{\max} \tag{3.13}$$

$$\iff r_{\max} - 1 \leq \sum_{i<v} \tfrac{lo_i + \sum_j c_j \cdot M_{ij}}{up_i + lo_i} \leq r_{\max} \tag{3.14}$$

$$\iff r_{\max} - 1 - \sum_{i<v} \tfrac{lo_i}{up_i + lo_i} \leq \sum_j c_j \cdot (\textstyle\sum_{i<v} \tfrac{M_{ij}}{up_i + lo_i}) \leq r_{\max} - \displaystyle\sum_{i<v} \tfrac{lo_i}{up_i + lo_i} \tag{3.15}$$

Notice that this optimization problem is actually a multidimensional 0-1 knapsack (that includes negative weights). Sadly, solving this problem is NP-Hard, and thus is probably not an time-efficient upper-bounding procedure. There are at least two ways to generate an upper bound for this problem, computable in polynomial time:

- Relaxing the constraint $c_j \in \{0,1\}$ into $c_j \in [0,1]$;
- Or relaxing the constraint (3.15) by removing it (note that it would also relax constraint (3.2c) on row v implicitly, by construction of the objective function that forces the equality with constraint (3.2b)).

Doing any of these actions will generate an upper bound for equation (3.11). We decide to go for the second possibility, as it greatly simplifies the computation and the objective function, which becomes:

$$\texttt{caseB}(v) := r_{\max} \cdot up_v + \sum_{i<v} \tfrac{lo_i \cdot (up_i - up_v)}{up_i + lo_i} + \max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \sum_{i<v} \tfrac{(up_i - up_v) \cdot M_{ij}}{up_i + lo_i} \tag{3.16}$$

$\min_v$ `caseB`(v) is an upper-bound for the original row-relaxed problem with $\sum_i r_i = r_{\max}$. One solution would be to enumerate all the possible values for v, which would lead to a computation time in $\mathcal{O}(nm^2 \log(n))$:

- for a given v, computing the first sum is in $\mathcal{O}(n)$
- the $\max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}}$ operator implies to compute n sums, each containing at most m elements, and then to sort these sums to select the greatest ones. Hence a complexity of $\mathcal{O}(nm \log(n))$;
- this must be repeated for each v, so at most n times.

However, `caseB`(v) is unimodular:

Theorem 3.2.5. *Let v^* be the optimal value for the problem described in equation* (3.11) *(with the cardinality constraint $0 \leq r_v \leq r_{\max}$). Then:*

- $\forall v \geq v^*: \ caseB(v) \leq caseB(v+1)$
- $\forall v \leq v^*: \ caseB(v) \leq caseB(v-1)$

caseB(v) is thus unimodular around v^*.

Proof. Let us prove the first claim, $\forall v \geq v^*: $ `caseB`$(v) \leq$ `caseB`$(v+1)$. Suppose that we have such a $v \geq v^*$. Let C_v^* be the optimum column selection found by `caseB`(v), and $\alpha = up_v - up_{v+1}$ (note that $\alpha \geq 0$ by construction). When we write r_i and r_i^*, they refer to the ones of

$\texttt{caseB}(v)$. Then:

$$\begin{aligned}
&\texttt{caseB}(v+1) \qquad (3.17)\\
&= r_{\max}\cdot up_{v+1} + \sum_{i\leq v}\tfrac{lo_i\cdot(up_i-up_{v+1})}{up_i+lo_i} + \max_{\substack{C\subseteq L_C\\ c_{\min}\leq|C|\leq c_{\max}}}\sum_{j\in C}\sum_{i\leq v}\tfrac{(up_i-up_{v+1})\cdot M_{ij}}{up_i+lo_i} \qquad (3.18)\\
&\geq r_{\max}\cdot up_{v+1} + \sum_{i\leq v}\tfrac{lo_i\cdot(up_i-up_{v+1})}{up_i+lo_i} + \sum_{j\in C^*_v}\Big(\sum_{i\leq v}\tfrac{(up_i-up_{v+1})\cdot M_{ij}}{up_i+lo_i}\Big) \qquad (3.19)\\
&\geq r_{\max}\cdot (up_v-\alpha) + \sum_{i\leq v}\tfrac{lo_i\cdot(up_i-up_v+\alpha)}{up_i+lo_i} + \sum_{j\in C^*_v}\Big(\sum_{i\leq v}\tfrac{(up_i-up_v+\alpha)\cdot M_{ij}}{up_i+lo_i}\Big) \qquad (3.20)\\
&\geq r_{\max}\cdot up_v + \sum_{i<v}\tfrac{lo_i\cdot(up_i-up_v)}{up_i+lo_i}\sum_{j\in C^*_v}\Big(\sum_{i<v}\tfrac{(up_i-up_v)\cdot M_{ij}}{up_i+lo_i}\Big)\\
&\quad - r_{\max}\cdot\alpha + \sum_{i\leq v}\tfrac{lo_i\cdot\alpha}{up_i+lo_i} + \sum_{j\in C^*_v}\Big(\sum_{i\leq v}\tfrac{\alpha\cdot M_{ij}}{up_i+lo_i}\Big) \qquad (3.21)\\
&\geq \texttt{caseB}(v) - r_{\max}\cdot\alpha + \sum_{i\leq v}\tfrac{lo_i\cdot\alpha}{up_i+lo_i} + \sum_{j\in C^*_v}\Big(\sum_{i\leq v}\tfrac{\alpha\cdot M_{ij}}{up_i+lo_i}\Big) \qquad (3.22)\\
&\geq \texttt{caseB}(v) + \alpha\Big(\sum_{i\leq v}\tfrac{lo_i+\sum_{j\in C^*_v}M_{ij}}{up_i+lo_i} - r_{\max}\Big) \qquad (3.23)\\
&\geq \texttt{caseB}(v) + \alpha\Big(\sum_{i\leq v} r_i - r_{\max}\Big) \qquad (3.24)\\
&\geq \texttt{caseB}(v) + \alpha\Big(r^*_v - r_v\Big) \qquad (3.25)
\end{aligned}$$

As we are above the optimum v^*, we have that $r_v \leq 0$. Moreover $0 \leq r^*_v \leq 1$. We obtain $\texttt{caseB}(v+1) \geq \texttt{caseB}(v)$. The proof is similar in the other direction. □

The unimodularity of $\texttt{caseB}(v)$ allows to compute the optimal v using a ternary search. This method has a total runtime of $\mathcal{O}(nm\log n\log m)$.

3.2.3 Case (C)

The mechanism for case (C) are similar to the ones of case (B). By doing the same relaxation, we find the following function to be an upper-bound for the row-relaxed problem with $\sum r_i = r_{\min}$ (with rows ordered by increasing lo_i):

$$\begin{aligned}\texttt{caseC}(v) := {} & (v + 1 - r_{\min}) \cdot lo_v + \sum_{i>v} \tfrac{lo_i \cdot (up_i - up_v)}{up_i + lo_i} \\ & + \max_{\substack{C \subseteq L_C \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \Big(\sum_{i \leq v} M_{ij} + \sum_{i>v} \tfrac{(up_i - up_v) \cdot M_{ij}}{up_i + lo_i} \Big) \end{aligned} \tag{3.26}$$

This function is also unimodular (proof omitted for conciseness; it is very similar from the proof of theorem 3.2.5). The time needed to compute $\texttt{caseC}(v)$ is in $\mathcal{O}(nm \log n \log m)$.

3.2.4 Wrapping up the upper bounds of C-MSS

An upper bound for each of the three cases of theorem 3.2.4 has been presented. An upper bound for the global can be derived, and is the minimum of these three case-specific upper bounds:

$$\min(\texttt{caseA}(), \min_v \texttt{caseB}(v), \min_v \texttt{caseC}(v)) \tag{3.27}$$

The total computation time is in $\mathcal{O}(nm \log n \log m)$, to be compared to the $\mathcal{O}(nm)$ time needed to compute the upper bound for the non-cardinality-constrained MSS.

The upper bounds from Theorem 3.2.1 and Equation (3.27) are not comparable:

Example 5. *For the matrix $\boldsymbol{M}_{ex}$ (see example 4) and constraints $r_{\min} = c_{\min} = 0$, $r_{\max} = 3$, $c_{\max} = 2$, we have that the bound from theorem 3.2.1 is 15, and the one from equation* (3.27) *is* $\simeq 16.92$. *For the same matrix, with $r_{\min} = c_{\min} = 2$, $r_{\max} = 6$, $c_{\max} = 3$, we have that the bound from theorem 3.2.1 is 31, and the one from equation* (3.27) *is* $\simeq 24.7$.

In the following sections, we present a branch and bound-based method that uses the upper bounds found above to trim the search space.

3.3 Adapting the upper-bound for partial solutions

We reuse the notations introduced in section 2.6. In order to use the upper bounds found in the previous section in a CP framework, we must adapt them to partial solutions.

For the non-cardinality constrained case, and case (A) of the bounds presented above, we obtain:

$$\texttt{caseA}() := \sum_{i \in R^{\perp}} \tfrac{up_i \cdot lo_i}{up_i + lo_i} + \max_{\substack{C^1 \subseteq C \subseteq C^1 \cup C^{\perp} \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \Big(\sum_{i \in R^1} M_{ij} + \sum_{i \in R^{\perp}} \tfrac{up_i \cdot M_{ij}}{up_i + lo_i} \Big) \tag{3.28}$$

For the case (B) of section 3.2.2, given that we order the rows in $R^\perp \cup R^1$ by decreasing up_i:

$$\texttt{caseB}(v) := (r_{\max} - |R^1|) \cdot up_v + \sum_{i<v \wedge i \notin R^1} \frac{lo_i \cdot (up_i - up_v)}{up_i + lo_i} + \max_{\substack{C^1 \subseteq C \subseteq C^1 \cup C^\perp \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \Big(\sum_{i \in R^1} M_{ij} + \sum_{i<v \wedge i \notin R^1} \frac{(up_i - up_v) \cdot M_{ij}}{up_i + lo_i} \Big) \tag{3.29}$$

This equation can be computed directly in $\mathcal{O}(nm \log m \log n)$ for each v. Using caching and trailing, we can however avoid recomputing some part of it and speed up the computation. Let us rewrite the equation with additional intermediate values:

$$\texttt{caseB}(v) := (r_{\max} - |R^1|) \cdot up_v \tag{3.30}$$

$$+ \overbrace{\sum_{i<v \wedge i \notin R^1} \frac{lo_i up_i}{up_i + lo_i}}^{\texttt{STa}(0,v-1)} - up_v \cdot \overbrace{\sum_{i<v \wedge i \notin R^1} \frac{lo_i}{up_i + lo_i}}^{\texttt{STb}(0,v-1)}$$

$$+ \max_{\substack{C^1 \subseteq C \subseteq C^1 \cup C^\perp \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \Big(\underbrace{\sum_{i \in R^1} M_{ij}}_{\texttt{included}_j} + \underbrace{\sum_{i<v \wedge i \notin R^1} \frac{up_i \cdot M_{ij}}{up_i + lo_i}}_{\texttt{STc}_j(0,v-1)} - up_v \cdot \underbrace{\sum_{i<v \wedge i \notin R^1} \frac{M_{ij}}{up_i + lo_i}}_{\texttt{STd}_j(0,v-1)} \Big) \tag{3.31}$$

We keep each sum above inside a dedicated reversible segment tree[De +97] ($2 + 2n$ segment tree of size m each), allowing computing in a very fast way ($\mathcal{O}(\log m)$) the needed sum. When a row i becomes (un)selected ($\perp \to 0/1$), the segment trees are updated such that their content in position i becomes 0, in $\mathcal{O}(\log m)$ each. Moreover, if the row is indeed selected, then the variables $\texttt{included}_j$ are increased with value M_{ij} for every column j. The whole update operation is thus in $\mathcal{O}(n \log m)$, and the computation of the bound for a specific v is in $\mathcal{O}(n \log(nm))$.

For the case (C) of section 3.2.3, given that we order the rows in $R^\perp$ by increasing lo_i and that we place before them the rows in R^1:

$$\texttt{caseC}(v) := (v + 1 - r_{\min}) \cdot lo_v + \sum_{i>v} \frac{lo_i \cdot (up_i - up_v)}{up_i + lo_i} + \max_{\substack{C^1 \subseteq C \subseteq C^1 \cup C^\perp \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C} \left(\sum_{i \leq v} M_{ij} + \sum_{i>v} \frac{(up_i - up_v) \cdot M_{ij}}{up_i + lo_i} \right) \tag{3.32}$$

The best upper bound for the C-MSS can be found using:

$$\min(\texttt{caseA}(), \min_{v \leq r_{\max} - |R^1|} \texttt{caseB}(v), \min_{|R^1| \leq v} \texttt{caseC}(v)) \tag{3.33}$$

The lowest of the upper bounds at any moment of the resolution is named $ub^{R^1, R^\perp, C^1, C^\perp}$. As shown earlier, these bounds can be computed in $\mathcal{O}(nm \log(m) \log(n))$, this being made at each node of the search tree.

3.4 Incumbent solution update

Incumbent solutions can still be created at any new node, but only if we have enough already-selected rows or columns. We must respect the cardinality constraints while building the heuristic solution (which are still creating using the heuristic shown in section 2.6.2).

Algorithm 7 Maintain incumbent solution

```
rowVal[$i$] = $\sum_{j \in C^1} M_{ij}$ $\forall i \in L_C \setminus C^0$
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 ($\perp \to 0/1$)
    if row becomes selected then
        for all $j \in L_C \setminus C^0$ do        ▷ No need to do it for non-selectable columns
            colVal[$j$] ← colVal[$j$] + $M_{ij}$
    updateIncumbentSolution()

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

function updateIncumbentSolution()
    if $|R^1| \geq r_{\min}$ then
        solR ← $\max_{\substack{C^1 \subseteq C \subseteq C^1 \cup C^\perp \\ c_{\min} \leq |C| \leq c_{\max}}} \sum_{j \in C}$ colVal[$j$]
        incumbentSolution ← max(incumbentSolution, solR)
    if $|C^1| \geq c_{\min}$ then
        solC ← $\max_{\substack{R^1 \subseteq R \subseteq R^1 \cup R^\perp \\ r_{\min} \leq |R| \leq r_{\max}}} \sum_{i \in R}$ rowVal[$i$]
        incumbentSolution ← max(incumbentSolution, solC)
```

3.5 Experiments

We first show experiments on synthetic data, showing efficiency differences between the existing approaches. We then show possible usages of the row/column number limiting constraints on real data.

Synthetic data We compare our method for C-MSS against Gurobi (with the MIP model). We generate for this purpose matrices of different sizes. All matrices are filled by a Gaussian noise of parameters ($\mu = -0.1, \sigma = 1$). 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 CP-based methods run

		C-MSS	
Size	*p*	MIP	Ours
500	5%	**99.9%**	91.7%
500	30%	**99.9%**	88.2%
500	70%	**99.8%**	92.7%
1000	5%	47.6%	**98.0%**
1000	30%	65.5%	**91.0%**
1000	70%	**99.5%**	85.1%
2000	5%	31.6%	**98.2%**
2000	30%	29.8%	**98.5%**
2000	70%	21.7%	**100.0%**
3000	5%	14.8%	**100.0%**
3000	30%	16.7%	**100.0%**
3000	70%	21.6%	**100.0%**

Table 3.1: Mean ratio between the best found solution for the method and the best solution found by all methods over all matrices. Higher is better. The method with the best result is highlighted.

a Large Neighborhood Search (LNS, [Sha98]) with 0.2 seconds per run, with an adaptive relaxation rate. All the methods are stopped after 60 seconds.The constraints are $r_{\text{min}} = c_{\text{min}} = 50$, $r_{\text{max}} = c_{\text{max}} = 100$. The results are shown in Table 3.1.

IJCAI author-venue matrix We extracted from DBLP [Ley09] the author/venue graph of the last five IJCAI conferences. The matrix is a 6777x2177 (authors, venues) with the number of articles presented by the authors in each venue. We exclude IJCAI from the matrix as it is, by construction, overrepresented.

In order to find a relevant, small community of authors and the venues to which they collectively participate a lot, we add the following (arbitrary) constraints:

- At most twenty authors should be selected. The number of venues must be between two and seven.
- We substract 1 from all entries in the matrix to force selecting authors who contribute to all the selected venues, rather than authors who contribute a lot to a single venue and not to the others.

Table 3.2 contains the approximate MSS until convergence by the LNS search. The authors believe it is optimal but were not able to close the problem due to the size of the matrix.

	SIGIR	CIKM	WWW	WSDM	AAAI	KDD
Zhaochun Ren	3	2	2	7	2	1
Fuzheng Zhang	0	1	6	3	1	3
Martin Ester	0	5	5	2	2	2
Wenwu Zhu	0	0	0	0	6	7
Min Zhang	6	3	3	2	0	0
Xuanhui Wang	3	2	3	7	0	1
Xiangnan He	8	0	3	1	1	0
Min Wang	1	4	2	3	0	5
Quan Yuan	4	7	2	1	1	5
Shaoping Ma	7	3	4	4	0	0
Peng Cui	0	0	0	0	6	9
Nemanja Djuric	2	1	8	2	0	2
Yu Zhang	0	0	0	0	8	5
Mihajlo Grbovic	3	1	6	4	0	3
Jun Yan	2	8	3	2	4	0
Michael Bendersky	6	2	1	6	0	1
Marc Najork	3	2	2	7	0	2
Dou Shen	4	3	1	2	5	2
Liangda Li	1	2	5	1	2	2
Donald Metzler	8	5	1	5	0	0

processing
computing
multimedia
natural
artificial
media
advertising
bioinformatics
retrieval
modeling
machine
learning
database
education
networking
algorithms
system
online
web
self-driving
deep
graph
mining
things
nlp
big
bitcoin
knowledge
user
internet
data
streaming
text
behavior
vehicles
management
information
distributed
social
intelligence
recommendation
recommender
language
systems
computational
search

Table 3.2: left: approximate C-MSS for IJCAI author/venue graph with at most twenty authors and between two and seven venues, with one unit removed from each edge. Authors and venues are in no particular order. Right: Word cloud from the topics indicated by the authors (from Table 3.2) on their Google Scholar profiles.

Interestingly enough, the C-MSS is a sub-community of the Information Retrieval/Data mining community, as emphasized by the choice of venues. This analysis is further reinforced with the observation of the Google Scholar profiles of the authors, shown as a word cloud on the right of Table 3.2. Some authors are "false positives", such as Wenwu Zhu or Yu Zhang, which are not part of this community, but are nevertheless selected as they published at lot to AAAI and KDD.

Migration data MSS can be used as a way to summarize information as illustrated next by analyzing a human bilateral migration matrix from [The18]. For every pair of countries, this matrix contains an approximation of the number of people having migrated from one country to another during the year 2017. We create a submatrix by keeping only OECD members on the destination countries (columns) and non-OECD members as source countries (rows). Thus, it can be interpreted as a bipartite graph of migration between non-OECD toward OECD countries. Finding a C-MSS on this bipartite graph will then produce a two group of countries, which, together, contain the maximum number of migrants given the size of the groups. This C-MSS can then be represented in other formats in order to be interpreted. We propose as an example Figure 3.1 (page 62), which is a Circos [Krz+09] plot representing the C-MSS of this matrix, with the additional constraint that at most eight members of the OECD, and a most eight non-members, are selected.

3.6 Chapter conclusion

We extended the definition of the Maximum Sum Submatrix problem with cardinalities on the rows and the columns. The problem becomes more complex, but the bounds and methods discussed in chapter 2 can be extended as shown in this chapter, while still giving good results against MIP methods.

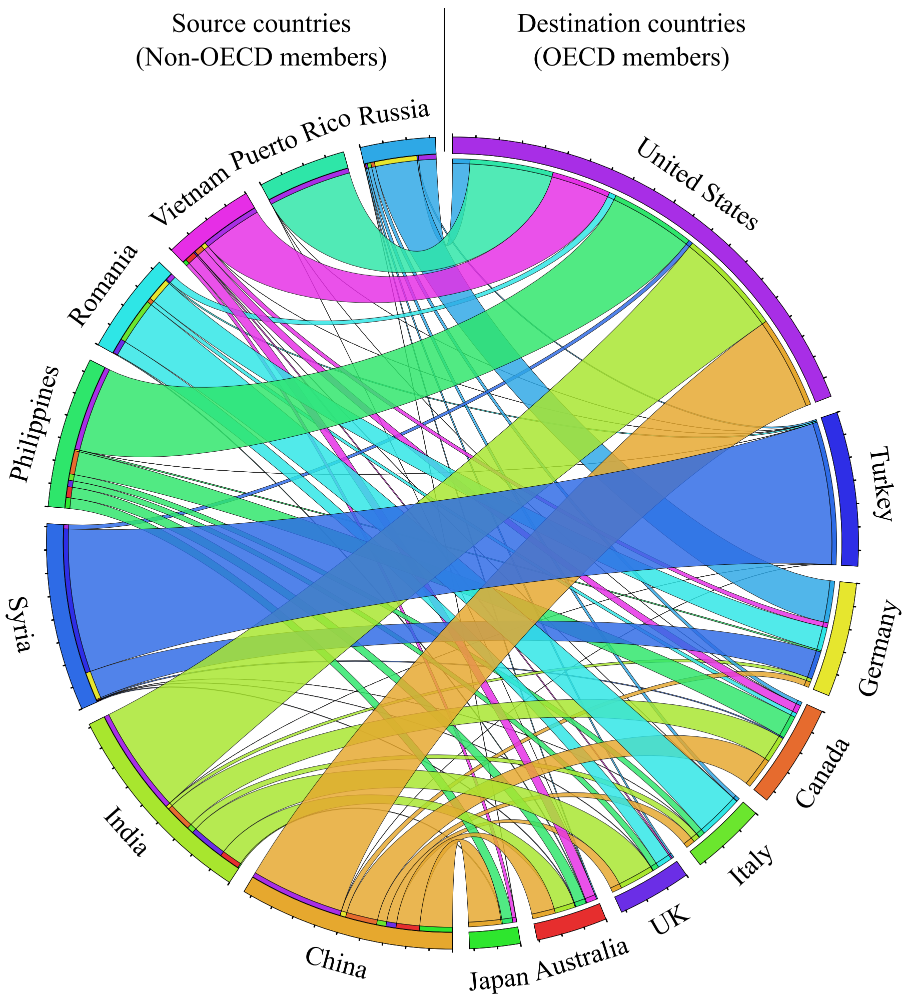


Figure 3.1: Circos plot of the C-MSS (max 8 rows, max 8 columns) of the modified migration matrix. OECD countries are on the left part of the plot, non-OECD ones on the right. Each tick on the outer circle represents half a million people.

Part II

Mining multiple submatrices

Mining a Set of Overlapping Submatrices | 4

This chapter is largely based on the paper G. Derval, V. Branders, P. Dupont, and P. Schaus. "The Maximum Weighted Submatrix Coverage Problem: A CP Approach". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by L.-M. Rousseau and K. Stergiou. Cham: Springer International Publishing, 2019, pp. 258–274. isbn: 978-3-030-19212-9. The two first authors made a similar amount of work on the article. This thesis' author's contribution involves mainly the creation of the algorithms to maintain the various states (and the idea of maintaining them), along with most of the writing itself, while Vincent Branders focused on the experiments and their analysis, and gave the initial idea for the dominance rules.

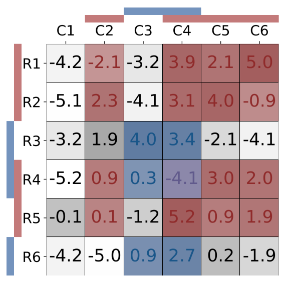
4.1 Introduction

Chapter 2 introduced the Maximum-Sum Submatrix (MSS) problem, which aims at finding submatrices (non-contiguous selections of rows and columns) with maximal sum.

The maximum weighted submatrix coverage problem (MWSCP), that we study in this chapter, generalizes the maximum-sum submatrix problem to K submatrices. Simply using the MSS problem multiple times would, of course, select K times the same matrix, we thus need to add additional constraints or account for this change in the objective function.

With the MWSCP we explore the second possibility, by not adding any constraint, thus allowing overlaps, but we modify the objective function such that if a cell is taken by multiple submatrices, it is only counted once in the objective function. That is, the aim is to maximize the weight of the union of the submatrices. An example is provided in Fig. 4.1.

Definition 7. The Maximum Weighted Submatrix Coverage Problem. *Given a matrix* $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ *and a parameter* K*, the maximum*



	C1	C2	C3	C4	C5	C6
R1	-4.2	-2.1	-3.2	3.9	2.1	5.0
R2	-5.1	2.3	-4.1	3.1	4.0	-0.9
R3	-3.2	1.9	4.0	3.4	-2.1	-4.1
R4	-5.2	0.9	0.3	-4.1	3.0	2.0
R5	-0.1	0.1	-1.2	5.2	0.9	1.9
R6	-4.2	-5.0	0.9	2.7	0.2	-1.9

The submatrix $(\{R1, R2, R4, R5\}; \{C2, C4, C5, C6\})$, in red, is of maximal sum as the value of the objective function is 27.3.

For $K = 2$, the two submatrices depicted in red, $(\{R1, R2, R4, R5\}; \{C2, C4, C5, C_6\})$, and blue, $(\{R3, R4, R6\}; \{C3, C4\})$, are of maximal sum. The objective value equals 38.6.

Figure 4.1: Example of matrix and associated submatrices of maximal sum.

weighted submatrix coverage problem is to select a set of submatrices $\{\boldsymbol{M}_{R_1,C_1}, \boldsymbol{M}_{R_2,C_2}, \ldots, \boldsymbol{M}_{R_K,C_K}\}$ *with* $R_k \in L_R$, $C_k \in L_C$ *for* $k = 1, \ldots, K$ *such that the sum of the cells covered by at least one submatrix is maximal:*

$$(R_1^*; C_1^*), \ldots, (R_K^*; C_K^*) = \underset{(R_1;C_1),\ldots,(R_K;C_K)}{\operatorname{argmax}} \sum_{i \in R, j \in C} \boldsymbol{M}_{i,j} \times \mathbb{1}_{cover}((i,j)) \tag{4.1}$$

where $\mathbb{1}_{cover}$ *is the indicator function over the set* $cover = \bigcup_{k \in 1 \ldots K} R_k \times C_k$.

The possibility of adding a constraint to disallow overlaps is studied in Chapter 5 with the Maximum-Weighted Set of Disjoint Submatrices (MWSDS).

4.1.1 Applications

The maximum weighted submatrix coverage problem has many practical data mining applications where one is interested to discover K strong relations between two groups of variables (rows and columns) represented as a matrix:

- In gene expression analysis, rows correspond to genes and columns to samples and the value in $M_{i,j}$ is the measurement of the expression of gene i in sample j. One is typically interested in finding subsets of genes that present high expression in a subset of the samples as it would indicate that a particular biological pathway made of these genes is active in these samples.
- In migration data, value $M_{i,j}$ represents the number of persons that moved from location i to j. The goal is the to identify groups of locations that together migrate to other groups of locations.

- A sports journalist could also be interested in Olympic games to discover group of countries that together obtained similar strong performances on the same subset of sports. The matrix value $M_{i,j}$ then represents the number of medals obtained by the country i in sport j.
- Dendrograms and Sankey plots are standard visualization tools to represent relations. Unfortunately, those plots quickly suffer from cluttering for large matrices. The MWSCP can be used as a preliminary step to preselect submatrices that can then be analyzed more easily with those plots.

Chapter's contribution

Our contributions are:

- The introduction of the maximum weighted submatrix coverage problem (MWSCP) as a generalization of the maximal-sum submatrix problem.
- A CP approach for solving MWSCP including filtering, lower-bound, dominance rules, a variable heuristic, and a Large Neighborhood Search (LNS) strategy.
- An evaluation of the performances of the CP approach as compared to a greedy baseline approach (using the maximal-sum submatrix problem as a subroutine) and two mathematical programming models on synthetic and real datasets.

4.2 CP approach

We use a CP approach to solve this problem. We first define precisely the search space, then explore the heuristics and dominance rules that can be used, and then use all these components in a Large Neighborhood Search.

We reuse notation concerning set variables (and their partial solutions) introduced in previous chapters (notably in section 2.6, page 31): we use set variables for each row/column set of each submatrix.

- R_k^1 is the set of already selected rows for submatrix k
- R_k^0 is the set of rows that will never be selected for submatrix k in this partial solution
- $R_k^\perp$ is the set of rows for which we don't yet know their state.

with $R_k^1 \cup R_k^0 \cup R_k^\perp = L_R$. Any possible solution R_k for the selected rows of submatrix k must thus respect $R_k^1 \subseteq R_k \subseteq R_k^1 \cup R_k^\perp$ or equivalently $R_k^1 \subseteq R_k \subseteq L_R \setminus R_k^0$.

The same goes for the columns with C_k^1, C_k^0 and $C_k^\perp$. A set variable S is bound if $S^\perp = \emptyset$.

We also define $R_k^{1,+j}$ (resp. $R_k^{1,-j}$) as the subset of R_k^1 whose matrix value in column j is positive (resp. strictly negative):

$$R_k^{1,+j} = \{i \in R_k^1 \mid M_{i,j} \geq 0\} \qquad R_k^{1,-j} = \{i \in R_k^1 \mid M_{i,j} < 0\} \tag{4.2}$$

Similar notations hold for C_k, 0 and $\perp$. The sum of the elements in a given row i (resp. column j) and in a column (resp. row) set S is noted as:

$$\underset{\text{row } i}{\text{sum}}(S) = \sum_{j \in S} M_{i,j} \qquad \underset{\text{col } j}{\text{sum}}(S) = \sum_{i \in S} M_{i,j} \tag{4.3}$$

The set of cells selected by at least one submatrix is denoted Cover^1. The set of cells excluded by all submatrices is denoted Cover^0:

$$\text{Cover}^1 = \{(i,j) \mid \exists k : i \in R_k^1 \wedge j \in C_k^1\} \tag{4.4}$$

$$\text{Cover}^0 = \{(i,j) \mid \forall k : i \in R_k^0 \vee j \in C_k^0\} \tag{4.5}$$

The CP resolution is made via a Depth-First-Search (DFS) exploration. The following subsections discuss the search space, sketch the algorithm and its key components.

4.2.1 Search Space

As explained in [BSD17b], the search space of MWSCP with $K = 1$ can be limited to searching on a single dimension, for instance C_1. Indeed, the variable R_1 can be fixed optimally in polynomial time by a simple inspection argument: $\forall i \in R_1^\perp : \underset{\text{row } i}{\text{sum}}(C_1) > 0 \implies i \in R_1^1$.

For $K > 1$, once all the columns set variables are fixed ($C_k \; \forall k \in [1..K]$) it remains to decide for each row i and each submatrix k whether i should be part of R_k or not. Those K decisions per row does not enjoy the monotonicity or the anti-monotonicity properties as illustrated on the next example.

Example 6. *Let us consider $K = 2$ with column selection $C_1 = \{1,3\}$, $C_2 = \{2,3\}$. For the 1×3 input matrix $\boldsymbol{M} = [[2,2,-3]]$. Individually for each submatrix, the sum of entries that would be covered by selecting this row in both R_1 and R_2 would be negative (-1). But since weights of* *covered elements count only once, the value -3 is added only once and the objective value obtained is 1. Now consider the matrix $\boldsymbol{M} = [[-2, -2, 3]]$. Individually for each submatrix, the sum of entries that would be covered by selecting this row in both R_1 and R_2 would be positive (1). But since weights of covered elements count only once, the value 3 is added only once and the final objective value is -1.*

Actually, those K decisions per row cannot be optimally taken in polynomial time anymore as stated in Theorem 4.2.1. As a consequence, the CP search will have to branch both on the rows and columns variables rather than branching on the columns only.

Theorem 4.2.1. *For fixed variables C_k $\forall k \in [2..K]$, fixing optimally R_k $\forall k \in [1..K]$ is NP-Hard.*

Proof. We reduce the NP-Hard Set Cover Problem [Kar72] to our problem: Given a universe $U = \{1, \dots, n\}$ and a set $\{S_1, \dots, S_K\}$ of K subsets of U, the Set Cover Problem is to find the minimum number of sets such that their union covers the universe. We construct a matrix with a single row and $n + K$ columns. The unique row values of this matrix are given by the regular expression $[K+1]\{n\}[-1]\{K\}$ (value $K+1$ repeated n times followed by -1 repeated K times). The column variables are fixed to $C_k = S_k \cup \{n+k\}$. In this reduction, S_k is selected if and only if $R_k = \{1\}$ for every set k. A first observation is that any optimal solution covers the universe otherwise it could be improved by K by selecting any additional set that contains an uncovered element. The optimal objective function can thus be written as $n \cdot (K+1) - |\{k \mid R_k = \{1\}\}|$. As $n \cdot (K+1)$ is fixed, maximizing this expression amounts at minimizing $|\{k \mid R_k = \{1\}\}|$ which is exactly the set cover objective. □

4.2.2 Resolution via Depth-First-Search

The CP resolution through Depth First Search (DFS) exploration is sketched in Algorithm 8. All the procedures are assumed to take the decision variables $\{R_1, \dots, R_K, C_1, \dots, C_K\}$ and the input matrix $\boldsymbol{M}$ as parameters.

The procedure selectUnBoundSetVar chooses a not yet bound set variable among $\{R_1^\perp, \dots, R_K^\perp, C_1^\perp, \dots, C_K^\perp\}$. The subsequent line chooses for the selected row/column set of some submatrix k, the specific row/column i (among the possible ones) to be included on the left branch and to be excluded on the right branch. The explored search tree is thus binary. Once the constraint is posted, and the previous state saved for later backtracking, the procedure propagateDominanceRule can include (exclude) rows or columns in every submatrix that can be proven to (not) participate in

Algorithm 8 Sketch of the DFS resolution algorithm

```
function SolveDFS( )
    if !allVariablesBound( ) then
        $S \leftarrow$ selectUnBoundSetVar( )
        $i \leftarrow$ selectValue($S^{\perp}$)
        for action $\in$ [set($i \in S^1$), set($i \in S^0$)] do
            saveState( )
            post(action)
            propagateDominanceRule( )
            (lb, cb, ub) $\leftarrow$ updateBounds( )
            best $\leftarrow$ max(best, cb)
            if ub $>$ best then
                SolveDFS( )
            restoreState( )
```

any optimal solution. The updateBounds function updates and returns the lower, current and upper bounds for the state. New incumbent solutions are obtained by transforming the partial assignment into a complete feasible solution that excludes all rows/columns in $\perp$. If the new possible solution (cb) is better than the best value found so far (stored in variable best), the current state $(R_1^1, \ldots, R_K^1, C_1^1, \ldots, C_K^1)$ is a better solution and the value of the variable best (storing the best objective found so far) is updated (and the solution is logged). Once this is done, a check is made to ensure that there may still be a better solution below this tree node, by verifying that the upper bound is greater than the best objective value found so far; if that is the case, the DFS continues recursively. Once these steps are done, the state is backtracked and the next state visited.

Efficient backtracking is achieved through trailing, which is a state management strategy that facilitates the restoration of the computation state to an earlier version. Trailing enables the design of reversible objects. We refer to MiniCP [MSV18] for a detailed description of trail-based solvers and to [Sai+13] for a trailed based implementation of set domains with sparse-sets.

The following subsections are dedicated to the four main functions of our algorithm: selectUnBoundSetVar, selectValue, propagateDominanceRule and updateBounds.

4.2.3 Functions selectUnBoundSetVar and selectValue

selectUnBoundSetVar chooses, at each step of the DFS, the next (unbounded) row/column interval set S to branch on, while selectValue selects the value $v \in S^{\perp}$ to include/exclude from this set when branching. That is, when a pair (S, v) has been chosen, the DFS branches on the left, by setting $v \in S^1$, and on the right, by setting $v \in S^0$. The decision of the interval set and of the value are not done independently. To choose the next (set,value) pair to branch on, our algorithm maintains two (reversible) counters per row or column and per submatrix:

- $t^{\text{row}}_{k,i}$ contains the sum of cell values that will be immediately added to the objective value if row i is included in R_k:

$$t^{\text{row}}_{k,i} = \underset{\text{row } i}{\text{sum}} \left(\{ j \mid j \in C^1_k \ \wedge (i,j) \notin \text{Cover}^1 \} \right) \tag{4.6}$$

- $p^{\text{row}}_{k,i}$ contains the sum of positive values in the line i that *could* be taken by submatrix k, i.e. whose columns have not been excluded:

$$p^{\text{row}}_{k,i} = \underset{\text{row } i}{\text{sum}} \left(\{ j \mid j \in (C^1_k \cup C^{\perp}_k) \ \wedge (i,j) \notin \text{Cover}^1 \} \right) \tag{4.7}$$

$t^{\text{col}}_{k,j}$ and $p^{\text{col}}_{k,j}$ are defined similarly. The algorithm then selects the (submatrix, row) (or (submatrix, column)) pair (k,i) (or (k,j)) that maximizes $t^{\text{row}}_{k,i}$ (or $t^{\text{col}}_{k,j}$). Ties are broken by maximizing $p^{\text{row}}_{k,i}$ (or $p^{\text{col}}_{k,j}$). The selected interval set and value are then R_k and i (or C_k and j).

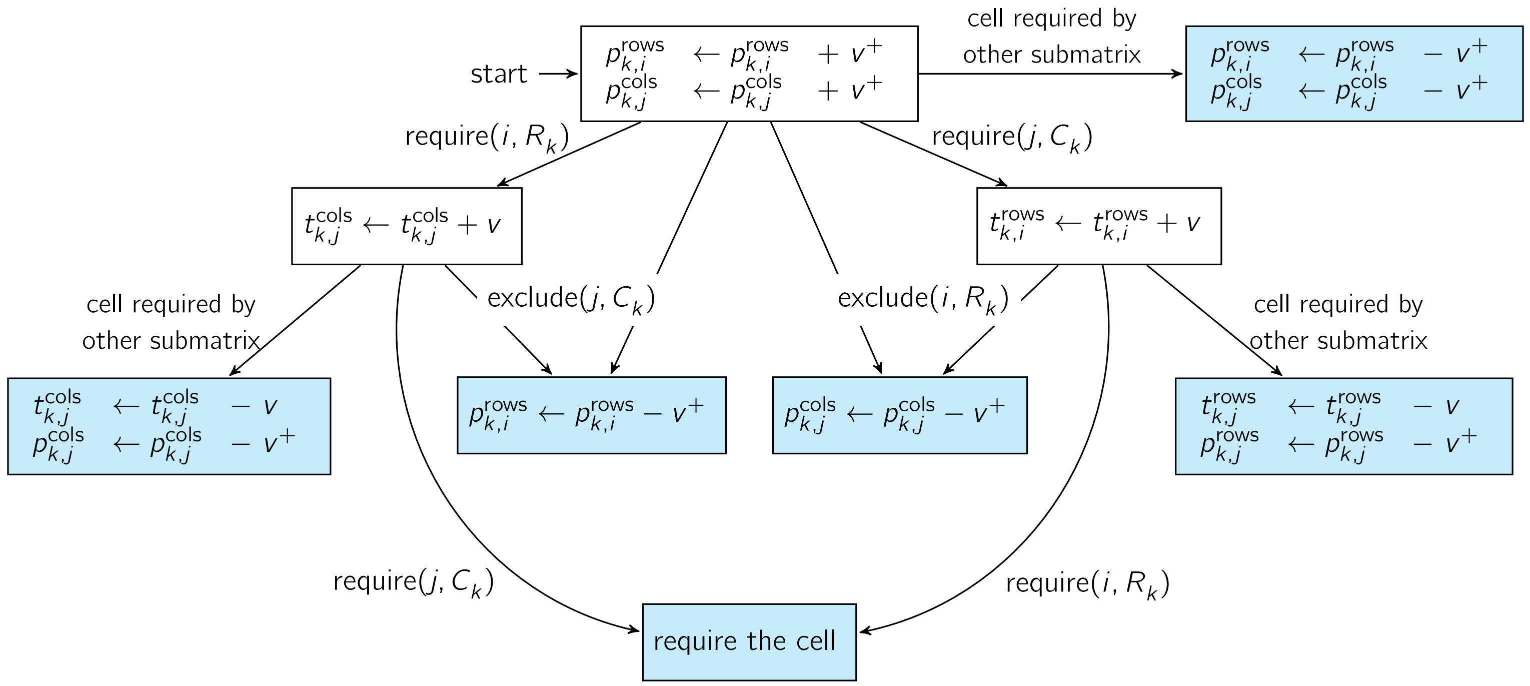

start
$p^{\text{rows}}_{k,i} \leftarrow p^{\text{rows}}_{k,i} + v^+$
$p^{\text{cols}}_{k,j} \leftarrow p^{\text{cols}}_{k,j} + v^+$
cell required by other submatrix
$p^{\text{rows}}_{k,i} \leftarrow p^{\text{rows}}_{k,i} - v^+$
$p^{\text{cols}}_{k,j} \leftarrow p^{\text{cols}}_{k,j} - v^+$
require(i, R_k)
require(j, C_k)
$t^{\text{cols}}_{k,j} \leftarrow t^{\text{cols}}_{k,j} + v$
$t^{\text{rows}}_{k,i} \leftarrow t^{\text{rows}}_{k,i} + v$
exclude(j, C_k)
exclude(i, R_k)
cell required by other submatrix
$t^{\text{cols}}_{k,j} \leftarrow t^{\text{cols}}_{k,j} - v$
$p^{\text{cols}}_{k,j} \leftarrow p^{\text{cols}}_{k,j} - v^+$
$p^{\text{rows}}_{k,i} \leftarrow p^{\text{rows}}_{k,i} - v^+$
$p^{\text{cols}}_{k,j} \leftarrow p^{\text{cols}}_{k,j} - v^+$
cell required by other submatrix
$t^{\text{rows}}_{k,j} \leftarrow t^{\text{rows}}_{k,j} - v$
$p^{\text{rows}}_{k,j} \leftarrow p^{\text{rows}}_{k,j} - v^+$
require(j, C_k)
require(i, R_k)
require the cell

Figure 4.2: FSM maintained for each (row, column, submatrix) i, j, k **in the variable/value selection algorithm. For simplicity,** $v = M_{i,j}$**,** $v^+ = \max(v, 0)$ **and** $v^- = \min(v, 0)$**. FSMs states in blue are terminal states.**

Recomputing these counters at each iteration is costly, as this operation runs in $\mathcal{O}(Knm + K(n+m))$ for the MWSCP with an $m \times n$ matrix and K submatrices. We propose here to maintain these counters using the finite state machine (FSM) shown in Fig. 4.2. The algorithm we propose virtually maintains a FSM for each (row, column, submatrix) triplet. The FSMs are updated each time a row/column is added to/excluded from a submatrix:

- When a row i is included in/removed from the submatrix k, at most n FSMs must be updated (one for each cell in the row).
- When a column j is included in/removed from the submatrix k, at most m FSMs must be updated (one for each cell in the column).
- Updating a cell is $\mathcal{O}(1)$, if it does not become *selected* by a submatrix (i.e. the row and column of the cell are both in the mandatory sets of the submatrix).
- If a cell becomes *selected*, $K-1$ other cells must be updated.

Given that $\Delta_{\text{rows}}, \Delta_{\text{cols}}$ and Δ_{selected} are respectively the number of added or excluded (submatrix, row) tables, added/excluded (submatrix, column) tables and selected cells between two calls of the algorithm, this update runs in $\mathcal{O}(\Delta_{\text{rows}} n + \Delta_{\text{cols}} m + \Delta_{\text{selected}} K)$. To this update process we must add the verification of the counters to select the best set/value pair, which is in $\mathcal{O}(K(m+n))$.

Over a complete branch of the DFS tree (which has a maximum depth of $K(m+n)$), we have that:

$$\sum_{\text{branch}} \Delta_{\text{rows}} \leq K \cdot m \quad \sum_{\text{branch}} \Delta_{\text{cols}} \leq K \cdot n \quad \sum_{\text{branch}} \Delta_{\text{selected}} \leq n \cdot m \quad (4.8)$$

Over a complete branch, the FSM-based algorithm maintains the states and returns the best set/value pair in $\mathcal{O}(K^2(m+n)^2)$, which is a significant improvement over the recomputation-based algorithm which runs in $\mathcal{O}(K^2(n^2 m + nm^2))$ over a complete branch.

4.2.4 Dominance rules

In some cases, given a partial assignment with some rows and columns already included in the set variables C_k and R_k, dominance rules permit to detect additional rows or columns that must be included in any optimal solution extending this partial assignment, or rows or columns that never participate in an optimal solution. The current state is defined by $(R_k^1, R_k^\perp, C_k^1, C_k^\perp)$, and we denote the optimal solution extending this state as $(R_k^{*\in}, \emptyset, C_k^{*\in}, \emptyset)$ with $R_k^1 \subseteq R_k^{*\in}$, $R_k^{*\in} \subseteq (R_k^1 \cup R_k^\perp)$, $C_k^1 \subseteq C_k^{*\in}$, $C_k^{*\in} \subseteq (C_k^1 \cup C_k^\perp)$.

Theorem 4.2.2 gives the condition to be satisfied to detect that a row i should be included in submatrix l in any optimal solution extending the current state.

Theorem 4.2.2.

$$\forall i \in R_l^{\perp} \ : \ \underset{row\ i}{\text{sum}}\left((C_l^1 \cup C_l^{\perp,-i}) \setminus (\bigcup_{k|k\neq l} C_k^{1,+i} \cup C_k^{\perp,+i}) \right) > 0 \Rightarrow i \in R_l^{*\in} \tag{4.9}$$

Proof. Let us assume the worst-case scenario: despite selecting all the columns with negative values in this row i, while other submatrices would take the columns with positive values, the submatrix still has a positive sum contribution for this row i.

I.e. there is no selection of columns (for this submatrix) and of rows/columns for the other submatrices such that this choice is not a net positive.

Therefore this row must be included in submatrix l in any optimal solution extending the current state. □

Theorem 4.2.3 gives the condition to be satisfied to detect that a row i will never be included submatrix l in any optimal solution extending the current state, using the best-case scenario.

Theorem 4.2.3.

$$\forall i \in R_l^{\perp} \ : \ \underset{row\ i}{\text{sum}}\left((C_l^1 \cup C_l^{\perp,+i}) \setminus (\bigcup_{k|k\neq j} C_k^{1,-i} \cup C_k^{\perp,-i}) \right) < 0 \Rightarrow j \notin R_l^{*\in} \tag{4.10}$$

These two properties (and their symmetric counterparts for columns) can be used in any node of the search tree to reduce the search space.

4.2.5 propagateDominanceRule: dominance rules check

Dominance rules from equations (4.9) and (4.10) (and their symmetric counterparts for the columns) can be used to reduce the search space. As in the previous subsections, recomputing the rules at each call to propagateDominanceRule is expensive ($\mathcal{O}(Kmn)$ at each call, $\mathcal{O}(K^2(m^2n + mn^2))$ over a complete branch of the DFS). We describe below how to maintain the rules on rows. Of course, the method is symmetric for columns.

As in selectUnBoundSetVar and selectValue, we maintain virtual FSMs for each triplet (row, column, submatrix), as shown in shown Fig.4.3. The FSMs collectively maintain two reversible values, shared between FSMs, for each (submatrix k, row i) table:

- $\mathrm{lb}_{k,i}$ is the value of the worst-case scenario for submatrix k and row i (the left part of equation (4.9))
- $\mathrm{ub}_{k,i}$ is the value of the best-case scenario for submatrix k and row i (the left part of equation (4.10))

The FSMs also maintain the number of *supports* of each cell (i, j), i.e. the number of submatrices that could still select the cell:

$$\mathrm{support}_{i,j} = \left|\{k \mid i \in (R_k^1 \cup R_k^\perp) \wedge j \in (C_k^1 \cup C_k^\perp)\}\right| \tag{4.11}$$

Each $\mathrm{support}_{i,j}$, shared across all FSMs, is maintained as reversible integer by the solver: its state can then be backtracked.

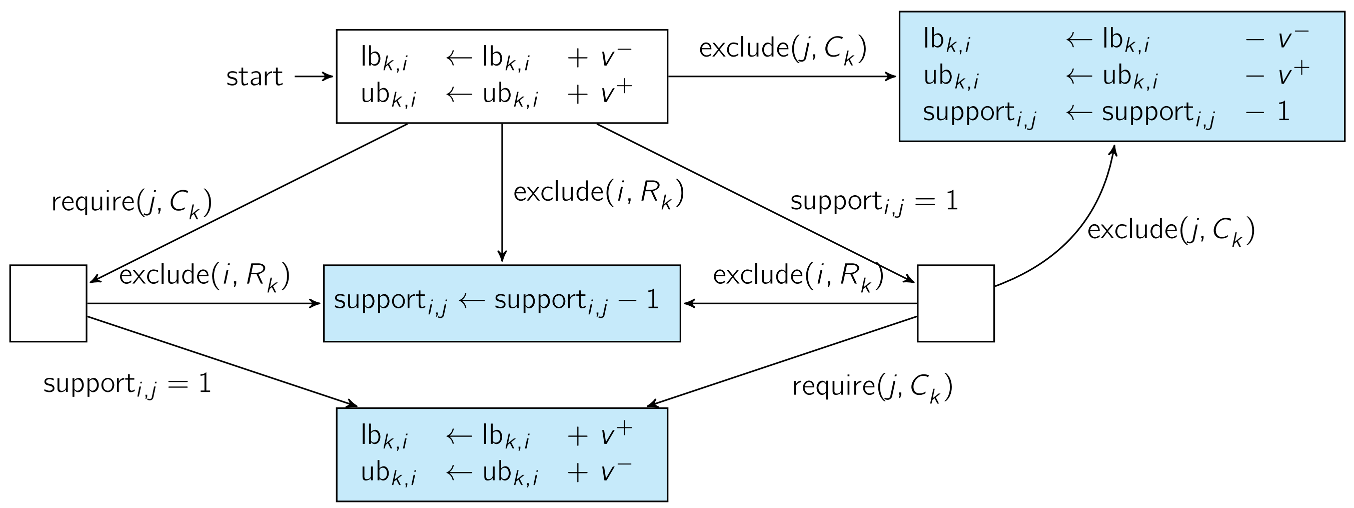

start
$\text{lb}_{k,i} \leftarrow \text{lb}_{k,i} + v^-$
$\text{ub}_{k,i} \leftarrow \text{ub}_{k,i} + v^+$
exclude(j, C_k)
$\text{lb}_{k,i} \leftarrow \text{lb}_{k,i} - v^-$
$\text{ub}_{k,i} \leftarrow \text{ub}_{k,i} - v^+$
$\text{support}_{i,j} \leftarrow \text{support}_{i,j} - 1$
require(j, C_k)
exclude(i, R_k)
$\text{support}_{i,j} = 1$
exclude(j, C_k)
exclude(i, R_k)
$\text{support}_{i,j} \leftarrow \text{support}_{i,j} - 1$
exclude(i, R_k)
$\text{support}_{i,j} = 1$
require(j, C_k)
$\text{lb}_{k,i} \leftarrow \text{lb}_{k,i} + v^+$
$\text{ub}_{k,i} \leftarrow \text{ub}_{k,i} + v^-$

Figure 4.3: FSM maintained for each (row, column, submatrix) i, j, k **in propagateDominanceRule. For simplicity,** $v = M_{i,j}$**,** $v^+ = \max(v, 0)$ **and** $v^- = \min(v, 0)$**. FSMs states in blue are terminal states.**

The transition and update operations of our FSMs are the following:

- When a row i (resp. column j) is excluded from a submatrix k, at most n (resp. m) cells' FSMs must be updated. The contribution of the cell (i, j) to $\mathrm{ub}_{k,i}$ and $\mathrm{lb}_{k,i}$ are removed and the support of the cell is decremented. Each of these operations are in constant time, and overall takes $\mathcal{O}(n)$ (resp. $\mathcal{O}(m)$).
- When a cell (i, j) becomes supported by only one remaining submatrix k ($\mathrm{support}_{i,j} = 1$), and the column j is included in this submatrix k ($j \in C^1_k$, and since $\mathrm{support}_{i,j} = 1$, it implies that $i \in (R^1_k \cup R^\perp_k)$), the value of lb and ub for this submatrix k is updated by the cell's value. This operation is also in constant time, and thus $\mathcal{O}(K)$ for all submatrices.
- When a row i (resp. column j) is included in a submatrix k, a check on all columns j (resp. rows i) must be performed to see if a cell (i, j) with $\mathrm{support}_{i,j} = 1$ and $i \in R^1_k$ and $j \in C^1_k$ exists. If that is the case, $lb_{k,i}$ and $ub_{k,i}$ are updated to include the value of the cell. Overall, this operation is $\mathcal{O}(n)$ (resp. $\mathcal{O}(m)$).

Once the update of the FSMs is done, each (row, submatrix) pair is verified w.r.t. the rules, in $\mathcal{O}(Km)$. A call to propagateDominanceRule is in $\mathcal{O}(Km + \Delta_{\mathrm{rows}} n + \Delta_{\mathrm{cols}} m + \Delta_{\mathrm{required}} K + \Delta_{\mathrm{support}=1} K)$. Over a complete branch, the number of operations required is in $\mathcal{O}(Km^2 + Kmn)$. If the rules are applied symmetrically on columns, the overall running time is in $\mathcal{O}(K \max(m, n)^2)$.

4.2.6 updateBounds: efficient lower and upper bounds computations

In order to run the Branch & Bound, upper bounds on the objective for the current tree node must be computed efficiently. The chosen method also provides a lower bound, with no additional (asymptotic) computational cost.

The upper bound ub is the sum of every cell that is either selected in a submatrix or that is positive and could still be selected. The lower bound lb is similarly defined, but keeping negative-valued cells. Formally, they are computed as follows:

$$\mathrm{ub} = \sum \{M_{i,j} \mid (i, j) \in \mathrm{Cover}^1 \vee (M_{i,j} > 0 \wedge (i, j) \notin \mathrm{Cover}^{\notin})\} \quad (4.12)$$

$$\mathrm{lb} = \sum \{M_{i,j} \mid (i, j) \in \mathrm{Cover}^1 \vee (M_{i,j} < 0 \wedge (i, j) \notin \mathrm{Cover}^{\notin})\} \quad (4.13)$$

Recomputing these bounds from scratch in each node is again costly: $\mathcal{O}(Knm)$. The running time can be improved by maintaining incrementally the number of submatrices supporting each cell, in the same way as previously done in propagateDominanceRule.

These bounds, stored as reversible floating point numbers, can then be maintained easily:

- When a row i is included in a submatrix k, check if any column j is already in C^1_k, and that $(i, j) \notin \text{Cover}^1$ yet. If that is the case and that $\mathcal{M}_{i,j} > 0$ (resp. < 0), increase ub (resp. lb) by $\mathcal{M}_{i,j}$. This operation runs in $\mathcal{O}(n)$.
- The similar operation must be performed when a column is included in a submatrix. Each of these operations runs in $\mathcal{O}(m)$.
- When a row i is excluded from a submatrix k, check if any column j is not already excluded ($j \notin (C^1_k \cup C^{\perp}_k)$). If that is the case, decrease $\text{support}_{i,j}$ by one. This operation runs in $\mathcal{O}(n)$.
- The same operation goes for excluded columns in $\mathcal{O}(m)$.
- When the $\text{support}_{i,j}$ is reduced to zero, if $M_{i,j} > 0$ (resp. < 0), then decrease ub (resp. lb) by $M_{i,j}$. This operation runs in $\mathcal{O}(1)$.

The whole maintenance process for the bounds behaves in $\mathcal{O}(\Delta_{rows} n + \Delta_{cols} m)$. Over a complete branch, the incremental method is in $\mathcal{O}(Knm)$, while the one based on recomputations is in $\mathcal{O}(K^2(n^2 m + nm^2))$.

4.2.7 Large Neighborhood Search

The exhaustive approach presented above eventually finds and proves the optimum value provided enough time is given. Unfortunately, the search space is so large that even for small matrices and a limited number of submatrices, it tends to quickly find a good solution but is not able to improve it. To overcome this limitation, we propose to embed the exhaustive CP search into a Large Neighborhood Search (LNS) [Sha98]. LNS is a local search approach using CP to discover improvements around the current best solution:

- First the CP exhaustive search is used during a limited time, to discover an initial solution.
- For a given number of iterations, the CP exhaustive search is used again but this time with some variables partially fixed (fragment) as in the current best solution.

In addition, to limit the risk of having an iteration stuck for too long, we limit the DFS to 1000 *failures*.

The current best solution at iteration t has the form $((R^{*\in}_{1,t}, \dots, R^{*\in}_{K,t});$ $(C^{*\in}_{1,t}, \dots, C^{*\in}_{K,t}))$ We propose three different fragment selection heuristics (part of the solution to constrain when restarting the LNS for next iteration):

1. Select uniformly at random a subset of rows and columns in the set of lines and columns used by some submatrix: $R^p \subseteq (\bigcup_{k \in M^p} R^{*\in}_{k,t})$, $C^p \subseteq (\bigcup_{k \in M^p} C^{*\in}_{k,t})$, then for each submatrix, include the set of rows and columns intersecting with those sets: $R^1_{k,t+1} = R^1_{k,t} \cap R^p$, $R^{\perp}_{k,t+1} = R \setminus R^1_{k,t+1}$ and similarly for columns.
2. A similar operator is defined with rows and columns selected inside the whole matrix: $R^p \subseteq R$, $C^p \subseteq C$. This allows for greater diversification, notably by allowing discovery of previously unselected rows/columns.
3. Selecting uniformly at random a subset of submatrices M^p for $p \in \{1, \dots, K\}$. For each of these submatrices, select at random different subsets of rows and columns $R^p_k \subseteq R^{*\in}_{k,t}$, $C^p_k \subseteq C^{*\in}_{k,t}$ that is constrained: $R^1_{k,t+1} = R^1_{k,t} \cap R^p_k$, $R^{\perp}_{k,t+1} = R \setminus R^1_{k,t+1}$ and similarly for columns.

Empirical observations show that these three operators are complementary.

4.3 Experiments

This section describes experiments conducted to assess the performances of the proposed algorithms and to provide guidance on the selection of the appropriate solution. We first evaluate the methods on synthetic datasets, where the optimum is known, then on real datasets.

We compare our exhaustive CP and LNS methods against a greedy baseline approach, CP-Greedy, that solves at each step the maximal-sum submatrix (K=1) problem using the CP approach from [BSD17b]. This approach iteratively selects the next best submatrix, on a modified matrix in which the previously selected entries are set to 0 such that there is no incentive to select several times the same (positive) entries. Each iteration is performed within $\frac{t^{\max}}{K}$ with $t^{\max}$ the allocated budget of time.

The implementation has been carried out on OscaR [Osc12], using Java 1.8.0 (Hotspot VM) on an AMD Bulldozer clocked at 2.1ăGHz; one core and 3 Go of RAM per instance.

The source code is available here: https://github.com/GuillaumeDerval/MWSCP.

4.3.1 Synthetic Datasets

A synthetic dataset composed of 1,617 instances have been generated using a Python script (available on Zenodo [Gui+18]). For those, the optimal solution is known[1] as they were all generated by implanting randomly K submatrices before adding some noise. Table 4.1 describes parameter values considered in the generation.The parameters used to generate the instances are described in Table 4.1.

Approaches are compared using any-time profiles as described in Definition 8.

Definition 8. Any-Time Profile. *Let $f(a, i, t)$ be the objective value of the best solution found so far by an algorithm a for an instance i at time t. Let $t^{\max}$ be the provided budget of time before interrupting a run. Let f_i^* be the optimal solution for i if known (as is the case for synthetic data). The any-time profile of a is the solution quality $Q_a(t)$ of a on all instances as a function of time:*

$$Q_a(t) = \frac{1}{|i|} \sum_i \frac{f(a, i, t)}{\max(f(a_i^*, i, t^{\max}), f^*)} \textit{ with } a_i^* = \underset{a}{\operatorname{argmax}} f(a, i, t^{\max}) \,. \quad (4.14)$$

Table 4.1: Parameters for the synthetic dataset generation

Parameter	Description	Values used
m, n	size of the matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$	$(800, 200)$, $(640, 250)$, $(400, 400)$
K	number of submatrices	2, 4, 8
o	minimum overlap between submatrices (in % of cells)	0, 0.3, 0.6
σ	background noise variance (mean is 0)	0, 0.5, 1.0
r, s	size of submatrices (noisy, Gaussian with $\sigma = \frac{r \text{ or } s}{20}$)	(35, 70), (50, 50)
seed	seed for matrix generation	$[0, 9]$

[1]Or at least a good approximation of it. Notice that the optimal solution may be slightly different than the implanted submatrices because of the noise addition.

	10s			20s			100s			1080s		
Parameters	GRE	EX	LNS	GRE	EX	LNS	GRE	EX	LNS	GRE	EX	LNS
$\{m = 400, n = 400\}$	**0.70**	0.33	0.37	0.74	0.57	**0.76**	0.76	0.75	**0.95**	0.77	0.75	**0.97**
$\{m = 640, n = 250\}$	**0.71**	0.34	0.32	0.75	0.48	**0.79**	0.77	0.74	**0.95**	0.77	0.75	**0.97**
$\{m = 800, n = 200\}$	**0.73**	0.34	0.29	**0.77**	0.48	0.61	0.79	0.77	**0.94**	0.79	0.78	**0.96**
$K = 2$	**0.85**	0.78	0.32	0.85	**0.88**	0.83	0.85	0.90	**0.96**	0.85	0.91	**0.97**
$K = 4$	**0.72**	0.20	0.30	**0.77**	0.51	0.72	0.78	0.74	**0.94**	0.78	0.75	**0.96**
$K = 8$	**0.57**	0.03	0.36	**0.64**	0.13	0.61	0.68	0.62	**0.94**	0.68	0.62	**0.97**
$o = 0\%$	**0.58**	0.27	0.34	0.67	0.45	**0.71**	0.71	0.66	**0.97**	0.71	0.66	**0.98**
$o = 30\%$	**0.71**	0.34	0.31	**0.73**	0.50	0.69	0.75	0.75	**0.93**	0.75	0.76	**0.95**
$o = 60\%$	**0.85**	0.40	0.34	**0.86**	0.57	0.77	0.86	0.86	**0.94**	0.86	0.86	**0.97**
$\sigma = 0.0$	0.73	0.34	**0.78**	0.78	0.63	**0.80**	0.81	0.77	**0.98**	0.81	0.78	**1.00**
$\sigma = 0.5$	**0.72**	0.33	0.04	**0.75**	0.44	0.67	0.78	0.74	**0.94**	0.78	0.74	**0.97**
$\sigma = 1.0$	**0.69**	0.33	0.16	**0.73**	0.44	0.68	0.73	0.75	**0.93**	0.73	0.75	**0.94**
$\{r = 50, s = 50\}$	**0.71**	0.34	0.34	**0.75**	0.52	0.73	0.77	0.76	**0.94**	0.77	0.77	**0.96**
$\{r = 35, s = 70\}$	**0.71**	0.32	0.32	**0.76**	0.50	0.71	0.78	0.75	**0.95**	0.78	0.75	**0.97**

Table 4.2: Synthetic dataset

$K = 4$		1s			5s			20s		
Type	Dataset	GRE	EX	LNS	GRE	EX	LNS	GRE	EX	LNS
migration	migration_0.001 [Dao+18a]	**0.96**	0.92	**0.96**	0.96	0.92	**0.99**	0.96	0.92	**1.00**
migration	migration_0.003 [Dao+18a]	0.87	0.89	**0.93**	0.87	0.89	**0.99**	0.87	0.89	**1.00**
migration	migration_0.005 [Dao+18a]	0.83	0.79	**0.96**	0.83	0.79	**1.00**	0.83	0.79	**1.00**
olympic	olympic_0.01 [IOC]	0.88	0.69	**0.92**	0.88	0.91	**0.97**	0.91	0.91	**1.00**
olympic	olympic_0.02 [IOC]	0.79	0.69	**0.87**	0.84	0.84	**0.97**	0.84	0.84	**1.00**
olympic	olympic_0.04 [IOC]	0.62	0.81	**0.91**	0.76	0.82	**0.96**	0.93	0.82	**1.00**
olympic	olympic_0.06 [IOC]	0.80	0.92	**0.93**	0.97	0.92	**0.98**	0.97	0.92	**0.99**
$K = 4$		10s			20s			100s		
Type	Dataset	GRE	EX	LNS	GRE	EX	LNS	GRE	EX	LNS
gene	alizadeh-2000-v1_095 [Sou+08]	**1.00**	0.48	0.82	**1.00**	0.48	0.82	**1.00**	0.48	0.92
gene	armstrong-2002-v1_095 [Sou+08]	0.73	0.60	**0.92**	0.73	0.60	**0.99**	0.73	0.60	**1.00**
gene	bhattacharjee-2001_095 [Sou+08]	0.82	0.31	**0.98**	0.91	0.86	**0.99**	0.91	0.96	**1.00**
gene	bittner-2000_095 [Sou+08]	**0.96**	0.53	0.86	0.96	0.53	**0.98**	0.96	0.53	**0.98**
gene	bredel-2005_095 [Sou+08]	0.98	0.86	**1.00**	0.98	0.86	**1.00**	0.98	0.86	**1.00**
gene	chen-2002_095 [Sou+08]	0.74	0.80	**1.00**	0.89	0.80	**1.00**	0.89	0.80	**1.00**
gene	chowdary-2006_095 [Sou+08]	0.82	0.83	**1.00**	0.82	0.83	**1.00**	0.87	0.83	**1.00**
gene	dyrskjot-2003_095 [Sou+08]	0.97	0.94	**0.99**	0.97	0.94	**1.00**	0.97	0.94	**1.00**
gene	garber-2001_095 [Sou+08]	**0.59**	0.24	0.58	**0.82**	0.32	0.58	**1.00**	0.50	0.86
gene	golub-1999-v1_095 [Sou+08]	0.86	0.88	**0.92**	0.86	0.88	**0.95**	0.86	0.88	**0.96**

Table 4.3: Real datasets

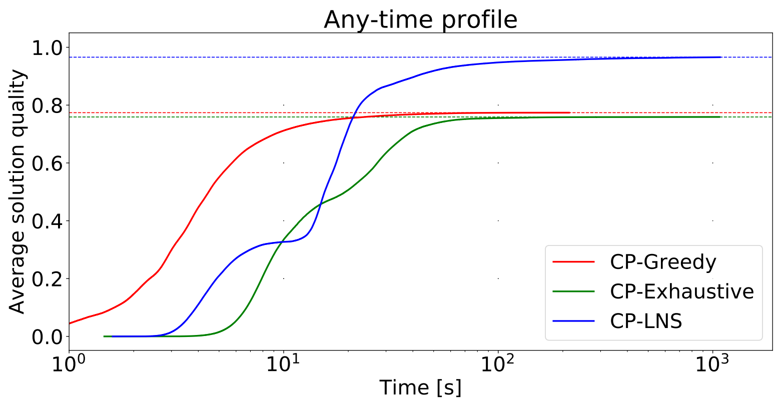


Figure 4.4: Comparison between CP-Greedy, CP-Exhaustive and CP-LNS on $1,617$ **matrices generated as described in section 4.3.1. The graph presents the any-time profile described in equation (8). For each instance, 18 minutes were allocated for computations.**

Fig. 4.4 gives the any-time profiles of the CP-Greedy baseline method, along with CP-Exhaustive (the exhaustive process presented above) and CP-LNS. The results clearly illustrates the overall better performances of the CP-LNS whenever the computation time exceeds roughly 20 seconds.

Table 4.2 presents, for each parameter value considered in the synthetic data generation, the performances of the algorithms. Reported performances are computed as the average performance of each algorithm obtained before a certain limit of computation time.

Through analysis of the performances with respect to parameters' values, we observed that the major parameters are, in decreasing order of influence, the following:

1. the degree of overlap between the submatrices
2. K, the number of submatrices

The difficulty of reaching good solution increases quickly as the minimum overlap parameter increases until 50%, after which it decreases. Similarly, as the number of implanted submatrices increases, it becomes increasinly harder to find good solutions.

4.3.2 Real Datasets

We also experiment with non-synthetic datasets of several types (*olympic*, *migration*, *genes*) described in section 4.1.1. The results, presented in Table 4.3, are similar to those obtained for synthetic datasets. CP-LNS is the best method on most datasets given 10 seconds of computation time, with two notable exceptions (alizadeh and garber datasets), in which case LNS did not find the optimum in the 20 minutes allowed for each dataset.

4.3.3 Comparison Against Mixed Integer Linearly and Quadratically Constrained Programming

We tested our methods against MIP (linear) and MIQCP (quadratic terms in the constraints) methods. As these two methods do not perform well on bigger instances, we do not integrate them in our experiments on large matrices, presented above.

$$
\begin{array}{rll|rll}
 & \text{MIP model} & & & \text{MIQCP model} & \\
\max & \sum_{i,j} M_{i,j} \cdot s_{i,j} & & \max & \sum_{i,j} M_{i,j} \cdot s_{i,j} & \\
s_{i,j} & \geq e_{i,j,k} & \forall i,j,k & K \cdot s_{i,j} & \geq \sum_k r_{k,i} \cdot c_{k,j} & \forall i,j \\
s_{i,j} & \leq \sum_k e_{i,j,k} & \forall i,j & s_{i,j} & \leq \sum_k r_{k,i} \cdot c_{k,j} & \forall i,j \\
e_{i,j,k}+1 & \geq r_{k,i} + c_{k,j} & \forall i,j,k & & & \\
2 \cdot e_{i,j,k} & \leq r_{k,i} + c_{k,j} & \forall i,j,k & & & \\
\end{array}
$$
$$\text{All variables} \in \{0,1\}$$

MIP and MIQCP methods are plagued by the number of variables, that is in $\mathcal{O}(Knm)$ for MIP and $\mathcal{O}(K(n+m))$ for MIQCP, and by the number of constraints, which is $\mathcal{O}(Knm)$ for MIP and $\mathcal{O}(nm)$ for MIQCP. Tables 4.4a and 4.4b show that both models are slow compared to our LNS method, and are heavily affected by matrix size, number of submatrices to find and noise. For bigger submatrices, such as the synthetic and real ones presented in the previous section, both methods timeout either without returning solutions or with comparatively poor solutions.

4.4 Chapter conclusion

We presented a generalization of the Maximal-Sum Submatrix Problem to multiple submatrices, called the Maximum Weighted Submatrix Coverage Problem (MWSCP), along with a method to solve this problem based on constraint programming and large neighborhood search. Experiments on both synthetic and real datasets show that our CP-LNS method finds consistently better solutions (when more than 10 seconds are allocated) than both MIP/MIQCP, an exhaustive CP method and a greedy approach using the method from [BSD17b].

Table 4.4: Comparison between CP-LNS, MIP and MIQCP, on a synthetic dataset (generated as described in section 4.3.1). All methods were given a fixed time limit of 300 seconds. The metric used is the any-time profile at the time limit (see definition 8). CP-LNS finds the optimum on each dataset. The time when the best found solution was found is indicated inside parentheses. Experiments made on Gurobi 8.1.0.

K	σ	CP-LNS		MIP		MIQCP	
2	0.0	**1.00**	(1s)	**1.00**	(**0s**)	**1.00**	(1s)
2	0.5	**1.00**	(**1s**)	**1.00**	(7s)	**1.00**	(7s)
2	1.0	**1.00**	(**1s**)	0.89	(233s)	0.79	(57s)
3	0.0	**1.00**	(2s)	**1.00**	(**1s**)	**1.00**	(2s)
3	0.5	**1.00**	(**3s**)	**1.00**	(140s)	**1.00**	(138s)
3	1.0	**1.00**	(**3s**)	0.74	(254s)	0.48	(256s)
4	0.0	**1.00**	(2s)	**1.00**	(**1s**)	**1.00**	(62s)
4	0.5	**1.00**	(**3s**)	**1.00**	(252s)	0.88	(290s)
4	1.0	**1.00**	(**6s**)	0.64	(260s)	0.69	(225s)
5	0.0	**1.00**	(**4s**)	**1.00**	(79s)	**1.00**	(275s)
5	0.5	**1.00**	(**5s**)	0.82	(257s)	0.69	(237s)
5	1.0	**1.00**	(**6s**)	0.77	(24s)	0.36	(38s)

(a) Varying number of submatrices and noise, with matrices of size 50×50 **and submatrices of size** 16×16.

m	σ	CP-LNS		MIP		MIQCP	
50	0.0	**1.00**	(**0s**)	**1.00**	(1s)	**1.00**	(3s)
50	0.5	**1.00**	(**1s**)	**1.00**	(5s)	**1.00**	(7s)
50	1.0	**1.00**	(**1s**)	0.95	(207s)	0.82	(204s)
100	0.0	**1.00**	(4s)	**1.00**	(**1s**)	1.00	(33s)
100	0.5	**1.00**	(**1s**)	0.86	(293s)	1.00	(45s)
100	1.0	**1.00**	(**3s**)	0.65	(269s)	0.82	(191s)
200	0.0	**1.00**	(17s)	**1.00**	(**8s**)	**1.00**	(135s)
200	0.5	**1.00**	(**21s**)	0.37	(191s)	3%	(81s)
200	1.0	**1.00**	(**6s**)	0%	(0s)	5%	(134s)
400	0.0	**1.00**	(**1s**)	**1.00**	(31s)	**1.00**	(54s)
400	0.5	**1.00**	(**1s**)	0%	(1s)	0%	(0s)
400	1.0	**1.00**	(**1s**)	0%	(1s)	4%	(301s)

(b) Varying size of the matrix and noise, with matrices of size $m \times m$ **and** $K = 2$ **submatrices of size** $\lfloor \frac{m}{3} \rfloor \times \lfloor \frac{m}{3} \rfloor$.

Mining a Maximum Weighted Set of Disjoint Submatrices 5

This chapter is largely based on the paper V. Branders, G. Derval, P. Schaus, and P. Dupont. "Mining a maximum weighted set of disjoint submatrices". In: *International Conference on Discovery Science*. Springer. 2019, pp. 18–28.. The two first authors made a similar amount of work on the article. This thesis' author's contribution involves finding the proof for the NP-completeness of a part of the problem (see below), the redistribution of the reduced-costs, along with some programming and guidance for the experiments.

5.1 Introduction

The identification of not one but K different submatrices is a natural extension to the maximum-sum submatrix problem. Without modification to the objective function, an optimal selection of K submatrices with a maximal sum of weights consists of selecting K times an identical submatrix, which is the one of maximal sum. As this provides no useful information, the aim being to find pairwise-different submatrices, the objective function must be modified and/or additional constraints need to be imposed.

The maximum weighted submatrix coverage problem (explored in the previous chapter) is one of such extensions to the identification of K possibly overlapping submatrices with maximal weight. It relies on a modification of the objective function such that covered matrix entries contribute precisely once to the objective. The downside of this modification is that it tends to favor overlaps on the negative matrix entries. Indeed, overlaps on the positive entries will not improve the objective value. On the opposite, overlapping submatrices share the penalty of selecting a negative matrix entry.

In this chapter, we consider an alternative extension to the identification of K submatrices, relying on an objective function computed as the sum of submatrix weights, and the explicit addition of disjunction constraints.

Definition 9. The Maximum Weighted Set of Disjoint Submatrices*: Given a matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ and K, a target number of submatrices. The maximum weighted set of disjoint submatrices problem is to select a set of K submatrices $\{\boldsymbol{M}_{R_1,C_1}, \boldsymbol{M}_{R_2,C_2}, \ldots, \boldsymbol{M}_{R_K,C_K}\}$, such that each matrix entry is covered by at most one submatrix and the weight of the covered entries is maximal:*

$$(R_1^*; C_1^*), \ldots, (R_K^*; C_K^*) = \underset{(R_1;C_1),\ldots,(R_K;C_K)}{\operatorname{argmax}} \sum_{k=1}^{K} \sum_{(i,j) \in R_k \times C_k} M_{ij} \quad (5.1)$$

$$s.t. \quad (R_k \times C_k) \cap (R_{k'} \times C_{k'}) = \emptyset \quad \forall k, k' \in \{1, \ldots, K\}, k \neq k' \quad (5.2)$$

Disjunction constraints (5.2) enforce that the intersection of entries covered by any pair of submatrices is empty. In other words, each matrix entry cannot be selected more than once[1]. We must stress that any submatrix pair may share rows or columns. The disjunction constraint only prevents submatrices from sharing both rows and columns at the same time.

As the MSS is $\mathcal{NP}$-hard, so is this generalization. This problem can efficiently be solved using a combination of column generation, using CP, and mixed integer linear programming (MIP).

The definition implies that the ordering of the submatrices is irrelevant.

It should be stressed again that the disjunction constraints are of importance. Indeed, they avoid the selection of irrelevantly identical submatrices. The redefinition of the objective function such as in the previous chapter leads to the unexpected behavior of avoiding signal overlaps, whereas overlaps on noise are favored. By reducing overlaps on the rows *or* the columns (but not both simultaneously) we avoid such behavior. Also, it eases the solution's interpretability by a domain expert. Such a structure of the solution is commonly called *nonoverlapping nonexclusive nonexhaustive* in the context of biclustering.

Chapter's contributions

Our contributions are:

[1] If we restrict to, say, maximum G overlap, an optimal solution consists of $\lceil K/G \rceil$ groups of (at most) G identical submatrices.

- The introduction of the maximum weighted set of disjoint submatrices problem (MWSDS) as a generalization of the max-sum submatrix (MSS) problem.
- A mathematical programming approach to solve the MWSDS.
- The formulation of the MWSDS as a mixed integer linear program relying on CP to produce relevant variables.
- An evaluation of the performances of these two alternatives and the benefit of the column generation as compared to a greedy baseline approach (using the max-sum submatrix problem as a subroutine) on synthetic datasets.

5.2 Constraint Programming Approaches

We propose an approach using column generation in an integer linear programming setting. The column generation (or subproblem solving) process is responsible for the identification of candidate submatrices that can be put into a weighted set of disjoint submatrices. The proposed approach relies on constraint programming (CP) to solve subproblems.

To avoid confusions between columns of the original matrix and *columns* corresponding to variables in the LP terminology, we will use the *subproblem solving* terms rather than *column generation*. Complementary, *columns* of the LP problem are referred to as LP-variables.

General notations. As previously, for each submatrix k, a set variable R_k (resp. C_k) is introduced to represent the possible selection of rows (resp. columns) in submatrix k. By an abuse of notation, in the context of a CP-based search, they also represent set variables with partial solutions, along with their declinations ($R_k^1, R_k^0, R_k^\perp$, and columns equivalent, see section 4.2 page 67 for details).

5.2.1 Search Space

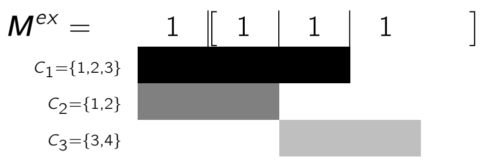
The search space of the maximal-sum submatrix problem (where $K = 1$) can be limited to searching on a single dimension, for instance the column set variable C_1, as explained in observation 1 page 17.

Indeed, the row set variable R_1 can be fixed optimally in polynomial time by a simple inspection argument:

$$\forall i \in L_R \colon \sum_{j \in R_1} M_{ij} > 0 \implies i \in R_1$$

For $K > 1$, once all the column set variables C_k are fixed, it remains to decide for each row i and each submatrix k whether i is to be selected ($i \in R_k$) or not ($i \notin R_k$). Example (7) illustrates the interdependence of these K decisions per row.

Example 7. *Given the following matrix and column set variables:*



Let us consider $K = 3$ with column set variables $C_1 = \{1, 2, 3\}$, $C_2 = \{1, 2\}$ and $C_3 = \{3, 4\}$, as shown above. There are $2^{K=3}$ possible ways of defining the row set variables R_1, R_2 and R_3.

It could be tempting to include the row in the first submatrix ($R_1 = \{1\}$). The optimal solution is $R_1 = \emptyset$, $R_2 = \{1\}$ and $R_3 = \{1\}$, however. Observe that row 1 cannot be included in all submatrices as overlaps are forbidden.

As previously with the MWSCP (chapter 4), these K decisions per row cannot be optimally taken in polynomial time anymore, as stated in Theorem 5.2.1. As a consequence, the search will have to assign both the row and column set variables, as opposed to the simpler $K = 1$ problem.

Let us define formally the MWSDS with fixed column selections:

$$R_1, \cdots, R_K = \underset{R_1,\cdots,R_K}{\operatorname{argmax}} \sum_{k=1}^{K} \sum_{(i,j)\in R_k \times C_k} M_{ij} \tag{5.3}$$

$$\text{s.t.} \quad (R_k \times C_k) \cap (R_{k'} \times C_{k'}) = \emptyset \quad \forall k, k' \in \{1, \ldots, K\}, k \neq k' \tag{5.4}$$

The notations are the same as in definition 9, but in this case the selections of columns for each submatrices (the C_i sets) are given.

Theorem 5.2.1. *The MWSDS with fixed column selections is $\mathcal{NP}$-Hard.*

Proof. We reduce the Maximum Weighted Independent Set (MWIS) problem to our problem. MWIS is NP-Hard (by immediate generalization from the Independent Set problem [GJ90]), and aims at finding, in a graph $G =< V, E >$ with weights w_v on each vertex $v \in V$, the set of vertices with the maximum sum such that they do not share edges in G. For simplicity, we represent edges and vertices as numbers: $V = \{1, \ldots, |V|\}$ and $E = \{1, \ldots, |E|\}$.

From an instance of the MWIS, we reduce it to an instance of MWSDS with fixed column selections. We create a matrix M of size $1 \times (|V| + |E|)$, such that

$$M_{1,i} = \begin{cases} w_i & \text{if } i \in \{1, \dots, |V|\} \\ 0 & \text{otherwise.} \end{cases}$$

The columns sets $C_1, \dots, C_{|V|}$ are constructed as follows:

$$C_v = \{v\} \cup \{|V| + e \mid e \in E \wedge \text{edge } e \text{ has } v \text{ as origin or destination}\}.$$

The interpretation is the following: each vertex in the graph G is transformed as a submatrix. If the single row of the matrix M is selected by a submatrix, then the vertex is included in the MWIS.

The non-overlapping constraint of MWSDS forbids two adjacent vertices (i.e. submatrices) to both be included in the solution (constructing an independent set), due to the way the column selections $C_1, \dots, C_{|V|}$ are constructed.

Resolving the MWSDS then leads to the same optimal objective result as the original MWIS problem, and the selected rows $R_v \; \forall v \in \{1, \dots, |V|\}$ indicates for each node v if the node is inside the MWIS ($R_v = \{1\}$) or not ($R_v = \emptyset$). As computing the MWIS in general graphs is $\mathcal{NP}$-Hard, and as the MWSDS with fixed column selections can encode the MWIS problem, we conclude that the MWSDS with fixed column selections is $\mathcal{NP}$-Hard. $\square$

5.2.2 Column Generation

Given a matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$, the number of possible rectangular submatrices is 2^{m+n} (there are some duplicates[2]). We consider the vector $\boldsymbol{v} \in \mathbb{R}^{mn}$ resulting from matrix $\boldsymbol{M}$ vectorization. Specifically, $\boldsymbol{v}$ is the $mn \times 1$ column vector obtained by stacking the columns of the matrix $\boldsymbol{M}$ on top of one another.

Let's represent a submatrix l as a column vector $\boldsymbol{s^l} \in \mathbb{B}^{mn}$ where $s^l_i = 1$ if entry v_i is covered by the submatrix l and $s^l_i = 0$ otherwise. The MWSDS is formulated using a $mn \times 2^{m+n}$ matrix $\boldsymbol{S}$ representing all possible submatrices[3]. A submatrix of the MWSDS is defined as a column

[2]There are multiple "empty" submatrices, taking some rows but no columns, or some columns but no rows. In practice there are precisely $2^{m+n} - 2^m - 2^n + 1$ different submatrices, but the difference is small and does not matter here, so we keep 2^{m+n} for simplicity.

[3]Note that we could represent each submatrix using only a boolean for each row/column rather than for each cell, making the matrix $\boldsymbol{S}$ sparser; we will see later in this chapter that the matrix is actually never built, and thus we keep this choice purely as it simplifies the reasoning and the notations.

$\boldsymbol{S}_{\cdot,l}$ of the $\boldsymbol{S}$ matrix. The weight w_l of submatrix l is the sum of its covered entries: $w_l = \sum_{i=1}^{v} v_i \cdot S_{i,l}$.

Example 8.

$$\boldsymbol{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad \boldsymbol{v} = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} & 1 & 1 & \\ & 1 & 0 & \\ \dots & 1 & 1 & \dots \\ & 1 & 0 & \\ & 0 & 1 & \\ & 0 & 0 & \end{bmatrix}$$

$\boldsymbol{v}$ is the vectorization of $\boldsymbol{M}$. $\boldsymbol{S}$ contains the representation of the possible submatrices of $\boldsymbol{M}$. The two example columns given here represent the submatrix with both rows and the two first columns, and the submatrix with only the first row but all columns.

The maximum weighted set of disjoint submatrices problem in equation (5.2) can be formulated as an Integer Linear program as follows:

$$\max_{\boldsymbol{x}} \quad \sum_{l \in L} w_l \cdot x_l \tag{5.5a}$$

$$\text{s.t.} \quad \sum_{l \in L} S_{i,l} \cdot x_l \leq 1 \qquad \forall i \in \{1, \dots, v\} \tag{5.5b}$$

$$\sum_{l \in L} x_l \leq K \tag{5.5c}$$

$$x_l \in \{0, 1\} \quad \forall l \in L \tag{5.5d}$$

where $L = \{1, \dots, 2^{m+n}\}$ denotes the set of possible submatrices. Equation (5.5b) ensures submatrices disjunction and equation (5.5c) enforces the selection of at most K submatrices. The decision variable x_l encodes the selection of submatrix l.

Defining the matrix $\boldsymbol{S}$ before solving the above ILP is not reasonable, even for small input matrices. Column generation, or subproblem solving, restricts the master problem, in equations (5.5a-5.5d), to a subset $L' \subseteq L$ of submatrices. Additional submatrices are progressively inserted in L', and the restricted master problem is solved until a provable optimal solution is found [BGN14].

Column generation, or subproblem solving, relies on iterations between solving the restricted master problem (RMP) and adding one or multiple submatrices. Candidate submatrices for addition to the RMP are submatrices that can improve the objective function of the LP relaxation upon addition to the restricted master problem. If no submatrix can improve the objective function, the optimal solution to the restricted master problem is an optimal solution to the full master problem.

We relax the integrality constraint on the RMP, and use its dual to find submatrices with a positive[4] reduced cost, i.e. that can improve the LP version of the RMP.

The dual of the master problem is given below.

$$\min_{\theta,\boldsymbol{\lambda}} \quad \theta \cdot K + \sum_{i=1}^{v} \lambda_i \tag{5.6a}$$

$$\text{s.t.} \quad \theta + \sum_{i=1}^{v} S_{i,l} \cdot \lambda_i \geq w_l \quad \forall l \in L \tag{5.6b}$$

$$\lambda_i \geq 0 \quad \forall i \in \{1, \dots, v\} \tag{5.6c}$$

$$\theta \geq 0 \tag{5.6d}$$

The dual values λ_i and θ correspond to the primal constraints defined in equation (5.5b) and equation (5.5c), respectively. Each "column" (thus, each *submatrix*) of the master problem is associated with a constraint in the dual (eq. (5.6b)). Values for λ and θ are obtained by solving an RMP.

Finding a submatrix with a positive reduced cost is called pricing. Such a submatrix is defined as any submatrix $l \in L$ for which

$$-\theta - \sum_{i=1}^{v} S_{i,l} \cdot \lambda_i + w_l < 0.$$

The LP version of the RMP is optimal if the pricing problem has no solution. Moreover, if the LP (being optimal) and the MIP version of the RMP have the same objective value, then the solution to the MIP is optimal.

While a pricing routine can return any submatrix with a positive reduced cost, the one with the maximum reduced cost is usually searched. The pricing problem can be reformulated as:

$$\sum_{i=1}^{mn} S_{i,l} \cdot (v_i - \lambda_i) > \theta$$

Solving this pricing problem is not trivial: it amounts to identifying a submatrix in the input matrix modified by the λ_i values such that its weight is larger than some θ. Given a modified input matrix, finding a submatrix of maximal sum using CP gives a solution to the subproblem, as long as the maximal sum submatrix is of weight larger than θ. We use the CP-with-global-constraint implementation (CPGC) provided in [BSD17a][5]. It is a

[4]Given that the problem is a maximization problem.

[5]Historically, the MSS chapter of this thesis has been written after this chapter, and the experiments were not rerun.

depth-first search approach composed of major CP ingredients: 1) filtering rules, 2) bounding procedure, 3) dominance rules, and 4) variable-value heuristic.

While the proposed method is not complete, using a branch-and-price algorithm [Sav97] is possible but non-trivial. The running time needed to solve the LP to optimality (i.e., to the point where no new subproblem with positive reduced cost exists) is quite high, as shown in the experiment section below.

The LP version of the RMP does not provide a boolean selection of submatrices. To effectively identify a solution to the original problem (MWSDS), the MIP version of the RMP, with integrality constraints enforced, is also solved for any solution to the ILP-RMP.

5.2.3 Avoiding redundancy

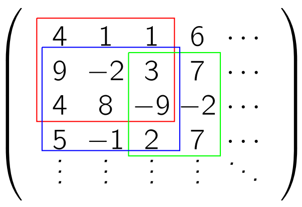
To maximize the information given by the dual values, we avoid having redundant constraints, notably the constraints (5.5b). For example, if we have two submatrices that overlap on more than one cell, we enforce only one constraint representing all the cells. Precisely, constraints (5.5b) are replaced by the following ones:

$$\sum_{l \in O} x_l \leq 1 \quad \forall O \in \Big\{ \{l \mid S_{i,l} = 1\} \;\Big|\; i \in \{1, \dots, v\} \Big\}. \tag{5.7}$$

That is, we enforce one non-overlap constraint per group of entries sharing the same intersecting submatrices (an *overlapping group*)[6]. We then redistribute the dual value of the constraint equally (we divide it by the number of entries) over all the entries in this overlapping group. This allows the method to avoid a pitfall of most MIP solvers: when facing multiple equivalent constraint, only one will be *tight*, i.e. having a non-zero dual value. This behavior leads to very high value for the λ_i on a very small number of entries. Redistributing the duals on all the entries in an overlapping group allows the subproblem solver to find more interesting submatrices.

Example 9. *Let us use this particular (part of) matrix, along with the submatrices in color (here they are contiguous for simplicity, but recall this is not mandatory):*

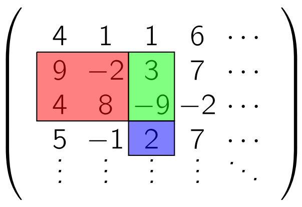
[6]Equation (5.7) uses the set notation to implicitly remove duplicates.



We could, as a naïve application of equation (5.5b)*, create a constraint for each cell overlapped by at least one submatrix, for a total of 15 here. Even if we remove those that are trivially true (overlapped by only one submatrix), 7 constraints remains.*

Even here, there are redundant constraints; take for example the four cells at the intersection of the second and third rows and the two first columns; they all intersect the same subset of submatrices.

In practice, the only non-trivial, non-redundant constraints are the one derived from the filled areas below:



Using the 7 original constraints is generally not a problem while solving the master problem; the solving time difference is generally small or negligible, the solver removing redundant constraints; however, as we put 7 constraints, we obtain 7 values for their reduced-cost. MIP solver generally ensure that only one redundant constraint is tight, and thus only one reduced-cost is positive.

These reduced-costs are interpreted in the pricing problem as values to be subtracted from a cell. Using directly the reduced-costs from these 7 redundant constraints would imply that only 3 cells get a value subtracted from their original value, and the other are left untouched. However, the reduced-costs associated with constraints can be shared between cells they cover, so these three assignations of reduced-costs are equivalent for the master problem:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 20 & 0 & 30 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 5 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 0 & 10 & 10 & 0 & \cdots \\ 10 & 0 & 20 & 0 & \cdots \\ 0 & 0 & 5 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 5 & 5 & 15 & 0 & \cdots \\ 5 & 5 & 15 & 0 & \cdots \\ 0 & 0 & 5 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

While they are equivalent for the master problem, they aren't for the pricing problem; distributing more evenly the values allows, in practice, to select better candidate submatrices (high values in a single cell simply forbids the cell, keeping the original problem nearly untouched).

Note that this trick doesn't modify correctness; it actually removes completely redundant constraints, which have the following properties:

- removing one of them do not modify the optimal objective value of the LP (primal or dual);
- the dual variables linked to these constraints share the same coefficient in the dual objective function, and the same coefficients in the dual constraints. Hence, given an optimal solution, subtracting a (constraint-preserving) value δ from one of these dual variable and adding it to another preserves the objective value. This technique is the basis of our redistribution trick;
- this technique can be used until all dual variables associated to the redundant constraint are equal to 0, but one whose dual value would be the sum of the previous dual values. The redundant constraints with a zero dual value can thus be removed safely.

5.2.4 Greedy Approach/Hot-Start

We define a greedy baseline approach that repeatedly solves the maximal weight submatrix ($K = 1$) problem, using the CP approach from [BSD17a]. The purpose of this approach is to provide a hot-start to the column generation approach.

The greedy approach iteratively selects the best next submatrix. Each time a new maximum submatrix is found, its content is replaced inside the main matrix by $-\infty$ (forbidding subsequent iterations from selecting these entries again).

In our experiments, each iteration is performed within $\frac{TO}{K}$ time where TO is the time-out for the greedy approach and K the number of submatrices that are searched for. If no solution has been found within the allocated time, the time-out is extended until at least one submatrix is found.

5.2.5 Mixed Integer Linear Programming

We propose a Mixed Integer Linear Programming model, based on a Big-M formulation of the problem. The model uses the binary variables r_i^k and c_j^k to represent the selection of row i and column j for submatrix k. The decision variables are used to compute the contribution of the row i for the submatrix k (p_i^k). The sum of the row contributions is the objective function to be maximized. The binary variables $s_{i,j}^k$ (one for each submatrix k and matrix cell i,j), indicating if cell i,j is covered by submatrix k, are used to ensure that selected submatrices are disjoint.

$$\max \quad \sum_{i\in L_R, k\in\{1,\dots,K\}} p_i^k \tag{5.8a}$$

$$\begin{aligned}
\text{s.t.} \quad p_i^k &\leq \sum_{j\in L_C} \left(M_{ij}\cdot c_j^k\right) + (r_i^k - 1)\cdot M_i^- && \forall i,k && (5.8\text{b})\\
p_i^k &\leq M_i^+ \cdot r_i^k && \forall i,k && (5.8\text{c})\\
2\cdot s_{i,j}^k &\leq r_i^k + c_j^k && \forall i,j,k && (5.8\text{d})\\
r_i^k + c_j^k &\leq 1 + s_{i,j}^k && \forall i,j,k && (5.8\text{e})\\
\sum_k s_{i,j}^k &\leq 1 && \forall i,j && (5.8\text{f})
\end{aligned}$$

where constants M_i^- and M_i^+ are respectively the lower bound and upper bound on the sum of row i's entries:

$$\forall i \in L_R\colon \quad M_i^- = \sum_{j\in L_C} \min(0, M_{ij}) \quad \text{and} \quad M_i^+ = \sum_{j\in L_C} \max(0, M_{ij}) \tag{5.9}$$

Constraints (5.8b) and (5.8c) ensures that the row contributions p_i^k are computed correctly. If $r_i^k = 0$, constraint (5.8c) ensure the contribution is zero, with the rhs of constraint (5.8c) being always positive. Otherwise ($r_i^k = 1$), constraints (5.8c) and (5.8b) become $p_i^k = \sum_{j\in L_C} M_{ij}\cdot c_j^k$, thus computing the effective value of the contribution.

Submatrices disjunction is ensured by constraints (5.8f) and relies on the variable $s_{i,j}^k$ which is computed as the product $r_i^k \cdot c_j^k$, linearized by eq. (5.8d) and (5.8e).

This model is plagued by the number of variables and constraints which are both in $O(mnK)$, mainly due to the non-overlap constraints.

5.3 Experiments

This section describes experiments conducted to assess the performances of the proposed algorithms and to provide guidance on the selection of the appropriate solution. Given enough time and memory, both the column generation approach and the MIP approach seem to converge to similar solution. Therefore comparing performances solely on the objective value of an approach is irrelevant. As a consequence, column generation and MIP approaches are evaluated and compared given a budget of time, the time-out TO, on synthetic datasets with implanted submatrices using any-time profiles:

Definition 10. Any-Time Profile. *Let $f(a, i, t)$ be the objective value of the best solution found so far by an algorithm a for an instance i at time t. Let $t^{\max}$ be the provided budget of time before breaking a run. The any-time profile of a is the solution quality $Q_a(t)$ of a on all instances as a function of time:*

$$Q_a(t) = \frac{1}{|i|} \sum_i \frac{f(a, i, t)}{f(a_i^*, i, t^{\max})} \text{ with } a_i^* = \operatorname*{argmax}_a f(a, i, t^{\max}) \,. \quad (5.10)$$

All experiments are performed using Java 1.8.0 on an AMD Bulldozer clocked at 2.1 GHz; one core and 6 GB of RAM per instance and a time-out TO of 2 hours. Solutions to the maximal sum submatrix problem are carried out on OscaR [Osc12].MIP and column generation approaches rely on Gurobi 8.1.0 [Gur18].

5.3.1 Combining Column Generation and Hot-Start

While the greedy approach cannot prove optimality, it may identify good solutions within short amounts of time. A natural way to take benefit from the greedy approach is to use it as a hot-start for the column generation. The greedy approach also serves as a subroutine to find solutions to generated subproblems (i.e. as a pricing problem solver).

In the subsequent experiments, all runs of the column generation are preceded by a greedy hot-start identifying up to K submatrices within the allocated time-out of 5 minutes. Afterward, the LP is solved, and the greedy approach starts on the matrix modified by the dual values (as explained in section 5.2.2) in order to find new submatrices that have a positive reduced cost in the LP RMP.

As this process generates more and more subproblems, the dual values of the RMP provide more guidance on the search for better subproblems. It is then more useful to seek multiple new subproblems later in the search process than at the beginning. Empirically, the subproblem generation is made such that, at the jth call to the subproblem generator, at most j new subproblems will be created. Given the non-trivial pricing problem, there is no guarantee that the subroutine identifies an optimal solution to the pricing problem.

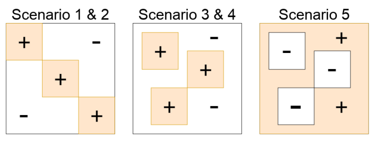
Note that all results of the CG method are computed from the MIP version of the RMP (with a binary selection of submatrices).

5.3.2 Datasets

This section presents the two datasets considered to evaluate performances of the column generation (CG) and mixed integer linear programming (MIP) approaches. We start with a dataset designed to evaluate the ability of the CG approach to refine solutions found by the greedy hot-start. The other dataset is constructed to evaluate the relative performances of the CG approach and the MIP approach.

5.3.2.1 Hot-Start Solution Improvement.

To evaluate the ability of the CG approach to improve over its hot-start, we designed five different scenarios where the greedy hot-start is bound to find a sub-optimal solution. The different scenarios are presented in Fig. 5.1. For scenarios 1, 2, 3 and 4, matrices are built to ensure that the maximum submatrix found by the greedy hot-start covers the whole matrix, thus giving very little information to the column generation method. Therefore, the benefit of the column generation is evaluated as the improvement of the objective value relatively to the objective value of the suboptimal hot-start solution.



Scenario	+ value	- value
1 and 3	$K+1$	-1
2 and 4	$\sim\mathcal{N}(K+1,\,1)$	$\sim\mathcal{N}(K+1,\,1)$
5	$\sim\mathcal{N}(2,\,2)$	$\sim\mathcal{N}(-1,\,1)$

Figure 5.1: Construction of matrices with suboptimal greedy solution. A matrix is defined by its positive and negative values. The presented scenarios rely on different structures of positive values insertions and the distribution of the positive and negative values defining the matrix.

For each scenario, 14 different matrices are generated, following the 14 combinations of matrix size and number of implanted submatrices indicated in Table 5.1. The performances of the hot-start and CG are then compared on the 70 matrices generated as described. As a complement of information, we also compare these results with the performances of the MIP approach.

Table 5.1: Combination of matrix size and number of implanted submatrices

	$K=2$	$K=5$	$K=8$	$K=10$	$K=20$
50×50	✓	✓			
100×100	✓	✓		✓	
200×200	✓	✓	✓	✓	✓
500×500	✓	✓		✓	✓

5.3.2.2 Relative Performances of CG and MIP.

A synthetic dataset composed of 720 instances has been generated using a Python script[7] following a similar protocol as in [Der+19b]. Data generation includes a normal background noise ($\sim \mathcal{N}(-1, 1)$) and K insertions of a submatrix with normal distribution ($\sim \mathcal{N}(1, 0.5)$)[8]. Rows and columns of each submatrix are uniformly selected from the set of rows and the set of columns of the matrix. Table 5.2 describes parameter values considered in the dataset generation.

It is worth stressing that given the allocated time and memory, MIP approach ends up without any solution and with memory issues for most (more than 75%) instances on larger matrices (800×200, 640×250, and 400×400). The parameter values in Table 5.2 are defined such that the MIP formulation can provide a solution to at least 75% of the 720 instances.

Table 5.2: Parameters for the synthetic dataset generation

Parameter	Description	Values used
m, n	size of the matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$	$(400, 100), (320, 125), (200, 200)$
K	number of submatrices	2, 5, 10
s	submatrix size (size $= (\frac{m \times s}{K}; \frac{n \times s}{K})$)	0.05, 0.01, 0.2, 0.5
seed	generation's seed	$[0, 19]$

5.3.3 Performances

Fig. 5.2 presents the any-time profile of the column generation, the greedy hot-start, and the mixed integer programming on the 70 synthetic in-

[7]The generation protocol with fixed seed is available on `https://www.dropbox.com/s/jh470v9etwg5js0/gen.py?dl=0`

[8]Notice that the optimal solution might be slightly different from the implanted submatrices because of the Gaussian distribution of the background and the implanted solutions.

stances described in section 5.3.2.1. These instances are generated such that a greedy approach is bound to obtain a low-quality solution.

The results clearly illustrate the benefit of the column generation approach on the hot-start: through the iterative definition of subproblems and identification of possibly improving submatrices, the CG approach can avoid the sub-optima reached by the hot-start. Results also illustrate the poor performances of the MIP approach, which, given roughly 25 times larger time-out than that of the greedy suboptimal hot-start, is outperformed by the latter approach.

Given that some sub-optimal solutions are efficiently found by the hot-start, the latter may complete the exploration of its search space before timing-out ($TO = 300$ seconds). As a consequence, the associated column generation process starts earlier and may improve over the hot-start solution before the hot-start times-out. It explains the shift between hot-start and CG curves before 300s.

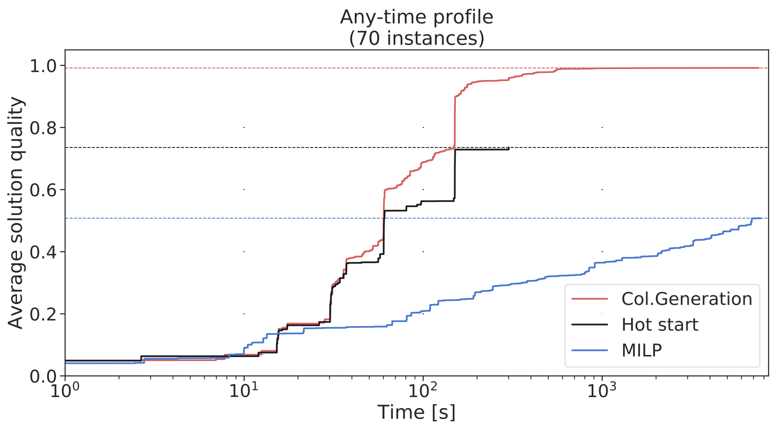


Figure 5.2: Comparison of the different methods proposed to solve the maximum weighted set of disjoint submatrices. The graph presents the any-time profile described in eq. (10). Results averaged on 70 **matrices generated as described in section 5.3.2.1 such that the greedy hot-start cannot find the optimal solution. For each instance, the time-out is fixed at 2 hours.**

Fig. 5.3 presents the any-time profile of the column generation and the mixed integer programming on the 720 instances generated as described in section 5.3.2.

These results illustrate the good performance of the column generation approach. MIP's worse performance is explained by the number of variables and constraints required to model the problem. On the smaller problems, with $K = 2$, the MIP approach appears to reach relatively good

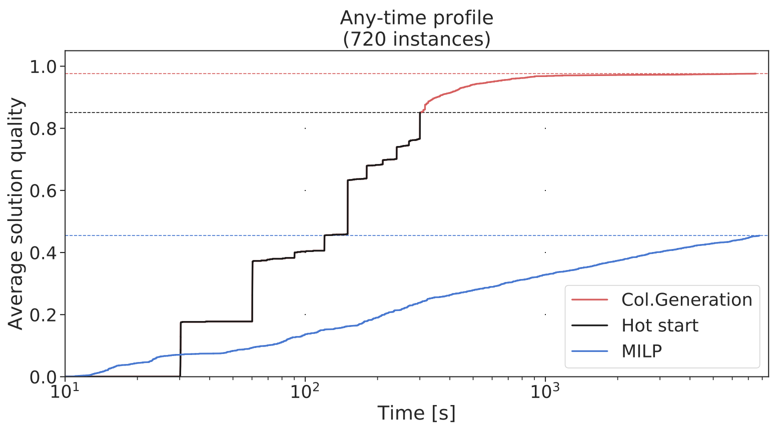


Figure 5.3: Comparison of the different methods proposed to solve the maximum weighted set of disjoint submatrices. The graph presents the any-time profile described in eq. (10). Results averaged on 720 **matrices generated as described in section 5.3.2. For each instance, the time-out is fixed at 2 hours.**

solutions. For larger values of parameter K, MIP performances do not reach a quarter of the column generation's average solution quality after 2 hours of computation time.

Each call to the greedy subroutine results in a single solution reported after some budget of time. This situation explains the sudden bumps in the hot-start: every $\frac{300}{K}$ seconds, a new solution is reported while in between, no improvements are reported.

5.4 Chapter conclusion

We explore the Maximum Weighted Set of Disjoint Submatrix Problem (MWSDS) and present two methods to solve it. One is based on mathematical programming and the other on constraint programming.

Our main contribution, a column generation method for the MWSDS, finds new candidate submatrices using dual variables of a linear relaxation of the submatrix selection problem. Experiments on synthetic datasets indicate that the column generation method consistently finds better solutions than the mixed integer linear programming approach.

The performances of the column generation can be further improved by complementing the exploration with a branch-and-price algorithm [Sav97]. Such improvement is non-trivial, however: the time taken to solve the underlying LP problem is already quite long, but is nonetheless an attractive direction for future work.

Part III

Future directions and conclusion

Discussion & future directions 6

In this chapter we succinctly describe some unexplored ideas, that may be the basis for future research, and discuss the results obtained and how they could maybe be improved.

6.1 MSS as a Bipartite Quadratic Pseudo Boolean Optimization problem

We assume some proficiency with maximum-flow algorithms and theory in this section.

The maximum sum subproblem can be represented as a Quadratic Pseudo Boolean Optimization (QPBO) problem; in this section we first define the QPBO problem, succinctly explore its properties and ways to solve it, then explain how we could use these properties and algorithms for the MSS and other problems seen in this thesis.

6.1.1 Quadratic Pseudo Boolean Optimization

Quadratic Pseudo Boolean Optimization (QPBO)[BH02; Rot+07; KR07] describes in the literature either a class of optimization problem or a method to solve them, depending on context.

Definition 11. Quadratic Pseudo-Boolean Optimization *(QPBO).*
QPBO aims at minimizing quadratic pseudo-boolean functions:

$$\min_{\boldsymbol{x}} f_{\boldsymbol{w}}(\boldsymbol{x}) = w^{\emptyset} + \sum_{i \in V} w^{i}_{x_i} + \sum_{(i,j) \in E} w^{i,j}_{x_i,x_j} \tag{6.1}$$

with $\boldsymbol{x} \in \{0,1\}^n$, $V = \{1,\dots,n\}$ (the nodes), $E \subseteq V \times V$ (the edges). $w^{\emptyset}$, $\boldsymbol{w}^i$ and $\boldsymbol{w}^{i,j}$ are respectively a constant, vectors of size 2, and matrices of size 2×2, giving a weight to each authorized assignation of variable/node/edge. Note that the edges are undirected: $w^{i,j}_{x_i,x_j} \equiv w^{j,i}_{x_j,x_i}$. The weights are grouped by an abuse of notation in the vector $\boldsymbol{w}$.

QPBO solvers generally assume that weights are either in $\mathbb{N}$ or $\mathbb{Z}$ (without loss of generality, see below) but they similarly can be in $\mathbb{R}$ or $\mathbb{R}^+$.

Intuitively, QPBO can be seen as a graph where a selection of nodes must be made. Each decision carries a specific weight: selecting a node has a weight, not selecting it has another weight, and every choice for every pair of variables may carry a weight too.

Example 10. *Here is an example, with four variables:*

$$w^{\emptyset} = 0$$

$$\begin{array}{rcc} & x_i{=}0 & x_i{=}1 \\ \boldsymbol{w}^1 = & 1 & -1 \\ \boldsymbol{w}^2 = & 2 & 0 \\ \boldsymbol{w}^3 = & -2 & 3 \\ \boldsymbol{w}^4 = & 4 & -2 \end{array}$$

$$\boldsymbol{w}^{1,2} = \begin{array}{cc|c} x_j{=}0 & x_j{=}1 & \\ 0 & 0 & x_i{=}0 \\ 1 & 0 & x_i{=}1 \end{array} \qquad \boldsymbol{w}^{1,3} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}\begin{array}{l} x_i{=}0 \\ x_i{=}1 \end{array}$$

$$\boldsymbol{w}^{2,4} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\begin{array}{l} x_i{=}0 \\ x_i{=}1 \end{array} \qquad \boldsymbol{w}^{3,4} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{array}{l} x_i{=}0 \\ x_i{=}1 \end{array}$$

The solution $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$ has an objective value of 4.

Most QPBO problem are displayed in *normal form*:

Definition 12. (QPBO normal form) *A QPBO instance is in normal form if:*

1. $\min(w^i_0, w^i_1) = 0\ \forall i \in V$
2. $\min(w^{i,j}_{v,0}, w^{i,j}_{v,1}) = 0\ \forall (i,j) \in E, v \in \{0,1\}$

A QPBO problem can be modified such that its value does not change for any vector $\boldsymbol{x}$. This is called a reparametrization:

Definition 13. *A* **reparametrization** $f_{\boldsymbol{w}'}$ *of a QPB function* $f_{\boldsymbol{w}}$ *is a QPB function such that* $f_{\boldsymbol{w}'}(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x})\ \forall \boldsymbol{x}$.

It is fairly straightforward to reparametrize a QPB function so that its weights are in normal form. If the first rule is not respected for a variable

i, i.e. $\delta = \min(w_0^i, w_1^i) \neq 0$, one can simply update the weights using the following operations:

$$\begin{aligned} w_0^i &\leftarrow w_0^i - \delta \\ w_1^i &\leftarrow w_1^i - \delta \\ w^{\emptyset} &\leftarrow w^{\emptyset} + \delta \end{aligned}$$

If the second rule is not respected for variables i, j and value v, the parameters can be updated using $\delta = \min(w_{v,0}^{i,j}, w_{v,1}^{i,j})$:

$$\begin{aligned} w_{v,0}^{i,j} &\leftarrow w_{v,0}^{i,j} - \delta \\ w_{v,1}^{i,j} &\leftarrow w_{v,1}^{i,j} - \delta \\ w_v^i &\leftarrow w_v^i + \delta \end{aligned}$$

These steps should then be repeated until the normal form is obtained.

It is easy to see that these reparametrizations do not change the value obtained by the function; we are merely moving weights around without modifying results. Once the QPB function is in normal form, the base weight $w_{\emptyset}$ is actually a lower-bound for the function.

Example 11. *Here is the normal form of the function shown in Example 10:*

$$w^{\emptyset} = -20$$

$$\begin{array}{rcc} & x_i{=}0 & x_i{=}1 \\ \boldsymbol{w}^1 = (& 1 & 0\ ) \\ \boldsymbol{w}^2 = (& 0 & 0\ ) \\ \boldsymbol{w}^3 = (& 0 & 5\ ) \\ \boldsymbol{w}^4 = (& 6 & 0\ ) \end{array}$$

$$\boldsymbol{w}^{1,2} = \begin{array}{c|c} \begin{array}{cc} x_j{=}0 & x_j{=}1 \end{array} & \\ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} & \begin{array}{c} x_i{=}0 \\ x_i{=}1 \end{array} \end{array}$$

$$\boldsymbol{w}^{1,3} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{array}{c} x_i{=}0 \\ x_i{=}1 \end{array}$$

$$\boldsymbol{w}^{2,4} = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \begin{array}{c} x_i{=}0 \\ x_i{=}1 \end{array}$$

$$\boldsymbol{w}^{3,4} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{array}{c} x_i{=}0 \\ x_i{=}1 \end{array}$$

The solution $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$ *still has an objective value of* 4.

6.1.2 The QPBO method

The QPBO method computes the objective value of a linear relaxation of the QPBO minimization problem here above. Let us first define the linear relaxation:

Definition 14 (QPBO, discrete Rhys form)**.** *[HHS84] An equivalent formulation of QPBO is as follows:*

$$\min_{\mathbf{x}} w^{\emptyset} + \sum_{i\in V}(x_i^1 w_1^i + x_i^0 w_0^i) + \sum_{(i,j)\in E,(a,b)\in\{0,1\}^2} (y_{i,j}^{a,b} w_{a,b}^{i,j}) \tag{6.2}$$

such that

$$\begin{aligned}
x_i^1 &= x_i && \forall i \in V && (6.3)\\
x_i^0 &= 1 - x_i && \forall i \in V && (6.4)\\
y_{i,j}^{a,b} &\geq x_i^a + x_j^b - 1 && \forall (i,j) \in E\ \forall (a,b) \in \{0,1\}^2 && (6.5)\\
y_{i,j}^{a,b} &\leq x_i^a, x_j^b && \forall (i,j) \in E\ \forall (a,b) \in \{0,1\}^2 && (6.6)\\
x_i &\in \{0,1\} && \forall i \in V && (6.7)\\
y_{i,j}^{a,b} &\in \{0,1\} && \forall (i,j) \in E\ \forall (a,b) \in \{0,1\}^2 && (6.8)
\end{aligned}$$

A linear version of this problem can be obtained by replacing constraints (6.7) *and* (6.8) *with the following ones:*

$$\begin{aligned}
x_i &\in [0,1] && \forall i \in V && (6.9)\\
y_{i,j}^{a,b} &\in [0,1] && \forall (i,j) \in E\ \forall (a,b) \in \{0,1\}^2 && (6.10)
\end{aligned}$$

To compute the optimal value of this linear relaxation, providing an upper bound, the QPBO method reparametrizes the weights. For that it uses a graph with special properties:

- A minimum cut/maximum flow provides the value for the linear relaxation optimum
- Augmenting paths created while computing the maximum flow can be seen as reparametrizations of the weights
- The minimum cut provides guarantees on the value of some part of the decision vector in some optimal solutions.

The graph is built as follows[BH02]:

- to each variable i is associated two nodes, x_i and $\bar{x}_i$, representing the variable and its negation;

$$\begin{tikzcd}
 & x_q \arrow[rr, "\frac{w^{p,q}_{0,0}}{2}"] \arrow[dd, bend left, "\frac{w^{p,q}_{1,0}}{2}"] \arrow[rrrd, bend left=40, "\frac{w^q_0}{2}"] & & \bar{x}_p \arrow[ll, bend left, "\frac{w^{p,q}_{1,1}}{2}"] \arrow[rd, "\frac{w^p_1}{2}"] \arrow[dd, bend left, "\frac{w^{p,q}_{1,0}}{2}"] & \\
s \arrow[ru, "\frac{w^q_1}{2}"] \arrow[rd, "\frac{w^p_1}{2}"'] \arrow[rrru, bend left=40, "\frac{w^p_0}{2}"] \arrow[rrrd, bend right=40, "\frac{w^q_0}{2}"'] & & & & t \\
 & x_p \arrow[uu, bend left, "\frac{w^{p,q}_{0,1}}{2}"] \arrow[rr, bend left, "\frac{w^{p,q}_{0,0}}{2}"] \arrow[rrru, bend right=40, "\frac{w^p_0}{2}"'] & & \bar{x}_q \arrow[ll, "\frac{w^{p,q}_{1,1}}{2}"] \arrow[uu, bend left, "\frac{w^{p,q}_{0,1}}{2}"] \arrow[ru, "\frac{w^q_1}{2}"'] &
\end{tikzcd}$$

Figure 6.1: QPBO method graph construction

- two special nodes, the source s and the sink t, are also present;
- for every non-null weight, two edges are added:

Weight	Edges to create		Edges' capacities
w^i_0	$x_i \to t$	$s \to \bar{x}_i$	$\frac{w^i_0}{2}$
w^i_1	$s \to x_i$	$\bar{x}_i \to t$	$\frac{w^i_1}{2}$
$w^{i,j}_{0,0}$	$x_i \to \bar{x}_j$	$x_j \to \bar{x}_i$	$\frac{w^{i,j}_{0,0}}{2}$
$w^{i,j}_{0,1}$	$x_i \to x_j$	$\bar{x}_j \to \bar{x}_i$	$\frac{w^{i,j}_{0,1}}{2}$
$w^{i,j}_{1,0}$	$x_j \to x_i$	$\bar{x}_i \to \bar{x}_j$	$\frac{w^{i,j}_{1,0}}{2}$
$w^{i,j}_{1,1}$	$\bar{x}_j \to x_i$	$\bar{x}_i \to x_j$	$\frac{w^{i,j}_{1,1}}{2}$

There is thus a one-to-one correspondence between weights (apart from $w_\emptyset$) and such a graph. Figure 6.1 shows a visual explanation of how the graph is built. By construction, it has a form of symmetry; for each edge $a \to b$ with weight c, there is an edge $\bar{b} \to \bar{a}$ with weight c (with $\bar{\bar{a}} = a$, $\bar{s} = t$ and $\bar{t} = s$).

The value of a minimum s-t cut inside this graph actually gives the objective value of the linear relaxation of the problem[Rot+07], and has additional interesting properties allowing to prune the domains:

Theorem 6.1.1 (Weak persistency)**.** *[Rot+07] Given the two sets of nodes S and T related to a minimum s-t cut (with $s \in S$ and $t \in T$), we have that there exists at least an optimal solution such that:*

$$x_i = 0 \text{ if } x_i \in S \wedge \bar{x}_i \in T \tag{6.11}$$

$$x_i = 1 \text{ if } \bar{x}_i \in S \wedge x_i \in T \tag{6.12}$$

the variables respecting at least one of those equations are said to be labeled by the s-t cut.

As there are multiple possible minimum cuts, different cuts can give a different number of *labeled* nodes; a maximum label can be obtained by computing strongly connected components on the residual graph (see [Rot+07] for the full algorithm). Similarly, it is possible to compute the minimum cut that labels the minimum number of variables: this labeling has the property that *all* optimal solutions have this partial labeling (*strong persistency*).

Computing the minimum s-t cut is usually done using a maximum-flow algorithm. Some algorithms are based on finding augmenting paths, i.e. path from the s to t in the residual graph with non-zero minimum weight w, then pushing this flow w through this path. This operation can actually be seen as a reparametrization of the QPB function[Rot+07][1].

6.1.3 MSS as a QPBO problem

A maximum-sum submatrix problem with a matrix $\boldsymbol{M}$ can be encoded as a QPBO problem by using as variables the rows r_i and the columns c_j, with the following weights:

$$\boldsymbol{w}^{r_i,c_j} = \begin{pmatrix} 0 & 0 \\ 0 & -M_{i,j} \end{pmatrix} \forall\, i \in L_R, j \in L_C$$

all the other weights being zero. The minimum obtained by the QPBO encoding is then minus the maximum sum of the original problem. A

[1]This breaks the "symmetry" property, but it can be restored easily by updating the edges' weights as follows: for each edge a and its mate b, do $w'_a \leftarrow \frac{w_a+w_b}{2}$, $w'_b \leftarrow \frac{w_a+w_b}{2}$

possible normal form can be:

$$
\begin{aligned}
w^{\emptyset} &= \sum_{i,j|M_{i,j}>0} -M_{i,j} \\
\boldsymbol{w}^{r_i} &= \begin{pmatrix} \sum_{j|M_{i,j}>0} M_{i,j} & 0 \end{pmatrix} \\
\boldsymbol{w}^{c_j} &= \begin{pmatrix} 0 & 0 \end{pmatrix} \\
\boldsymbol{w}^{r_i,c_j} &= \begin{cases} \begin{pmatrix} 0 & 0 \\ M_{i,j} & 0 \end{pmatrix} & \text{if } M_{i,j} \geq 0 \\ \begin{pmatrix} 0 & 0 \\ 0 & -M_{i,j} \end{pmatrix} & \text{if } M_{i,j} < 0 \end{cases}
\end{aligned}
$$

The MSS problem has a specific form: the variables r_i doesn't interact between themselves, i.e. they don't have edges between them in the QPBO graph. The same goes for the variables c_j. It is thus a bipartite quadratic pseudo-boolean problem, which has been studied (theoretically) in [PSK14].

The output of the QPBO method (the minimum cut value) is equivalent to the linear relaxation of the MSS.

Based on ongoing experimentations made while writing this thesis, using QPBO with efficient maximum flows algorithms (such as push-relabel-based algorithms) is faster than computing the LP bound using the Lagrangian (see section 2.4) by at least an order of magnitude.

6.1.4 Implications for other problems

Improvement on solving speed (and pruning) in the MSS may speed up the existing algorithms for the other problem presented in this thesis, notably for the Maximum Weighted Set of Disjoint Submatrices (MWSDS, see chapter 5), where finding a MSS is actually a subroutine (to generate a *column* in the column generation formulation).

For the CMSS (where the minimum and maximum number of selected rows/columns is limited), the implications are unclear; whether the QPBO graph is usable in this context or not is an open question. At least it provides a valid upper bound while not taking into account the cardinality constraints. The central concept of reparametrization must be adapted to account for the cardinalities, which does not seem trivial at first glance.

6.1.5 Ongoing work: a solver based on QPBO

There is apparently no complete solver based on QPBO in the literature. The closest existing match is solver of cost-function networks/WCSP such as Toulbar2[Hur+16; All+16; Coo+10]; however they use heuristics to compute a lower bound rather than solving exactly the linear approximation or rely on generic LP solvers[Coo+10].

We propose a simple architecture for a branch-and-bound QPBO solver, inspired by CP solvers such as OscaR[Osc12] and MiniCP[MSV18]. The main particularity of the solver would be that it maintains the underlying QPBO graph, notably during backtracking operations, and for this we propose the usage of trailing to maximize efficiency and minimize memory usage (see [Sch99] for a discussion between copying and trailing). The solver would thus maintain a variable for each edge of the residual graph.

Taking a decision implies that the graph must be updated, "forcing" a variable to respect the properties of equations (6.11) and (6.12). This can be done by using a reparametrization in linear time (see [Rot+07] page 12).

Such a solver would apply multiple times the maximum-flow computation (pruning nodes and domains) and the various pruning heuristics (such a the ones in chapter 2, but also with existing local heuristics for QPBO such a QPBOP[Rot+07]), until a fixed-point is reached where no further inference can be made. A listener system[MSV18] would allow the various heuristics to maintain their state during the computation.

Preliminary work (involving a functional solver implementing the ideas above) shows that there is an important speedup (multiple orders of magnitude) versus the method presented in Chapter 2, and empirically stress the importance of using trailing versus copying and of the choice of the maximum flow algorithm, with Push-relabel algorithms getting the best results.

6.2 Bounding heuristics as search heuristics

As a general rule, most experiments made during this thesis use a static branching strategy. However, most bounding methods presented assign a weight to each variable (generally its weight in the LP relaxation of the problem). These could be used as a search heuristic,to direct the search towards variables having a greater weight.

Experiments made during the thesis using this kind of ideas have generally been disappointing. We conjecture that always going in the direction of the best variable w.r.t. the LP relaxation actually doesn't allow the LP relaxation to give more accurate bounds.

6.3 Lagrangian-based bounds

The performance of the Lagrangian method in Chapter 2 is disappointing, as it is not able to compete with the Big-M-based bounds in practice. However, the implementation was made in plain Java, without any particular optimization. Exploring the opportunity of explicitly using modern vector CPU extensions, or even deep learning frameworks optimized for gradient descent and floating point computations could help make this method more competitive.

Moreover, aside this purely technical point, the way the subgradient descent algorithm is made can (and should probably) be changed. Adding less trivial techniques such as Nesterov momentum[Nes83] or modern optimizers would probably help to obtain a faster convergence.

6.4 Composability

The problems approached in this manuscript all (except C-MSS) have an important feature: each possible submatrix is (part of) a solution, that is, the entire space of submatrices is the search space. The bounds, reduced-cost filtering and other methods we derive in the previous chapters all rely on this fact. Sadly, it implies that some of the filtering we use is not composable: adding a new constraint will, in general, break the results obtained. Suffice to say, the length of chapter 3 where we add a cardinality constraint to the MSS is proof in itself that it is indeed not trivial.

Additional research in composability to make the known filtering algorithms (or others) work with different classes of constraints may prove beneficial for the end user.

6.5 Dual value redistribution in MWSDS

Again in Chapter 5, we presented a trick to reduce the number of redundant constraint in the RMP, reducing the number of dual variables. This trick made each constraint correspond to a group of cells intersecting the same submatrices rather than a single cell of the original matrix.

We chose during this thesis to redistribute equally the dual value on the cells it corresponds to. However, it is possible that alternative strategies may be beneficial, such as focusing the redistribution only on positive cells.

Conclusion | 7

In this thesis we explored four biclustering problems:

- the Maximum-Sum Submatrix problem, that aims at finding a submatrix whose elements sum is maximal. We introduced new bounds for the problem, and used them in a CP framework, along with dominance rules and reduced-cost filtering;
- A variant with cardinality constraints, C-MSS, to allow users to control the output more precisely. We adapted the bound of the MSS to account for these new constraints.
- the Maximum Weighted Submatrix Coverage Problem (MWSCP), to find multiple submatrices at once in datasets where the *solution* can be composed of multiple biclusters. We use a Finite-State-Machine based constraint to maintain the state of the bounds in a CP framework.
- the Maximum Weighted Set of Disjoint Submatrices (MWSDS), that forbids overlaps. A column generation scheme is used, which works by finding a single MSS at a time on a modified matrix, then solving a master problem with the found MSSs, before finding a new one, and until convergence.

The research on these problems was motivated by the analysis of gene expression matrices, that lists, for a series of *samples* (rows) and *genes* (columns), the expression (cells) of this gene in that sample. Submatrices with large sums represent subset of genes very active in large subsets of samples. Maximum-sum submatrices and adjacent techniques have proven useful in such contexts [BSD19], and we redirect to the thesis of Vincent Branders, co-author of some articles on which this thesis is based, for an in-depth analysis of the biological aspects of the usage of these methods [Bra21]. Examples have been shown in this thesis of other uses of the MSS family of techniques and problems. In practice, these data-mining tools can be used on any matrices whose content represents the force of a relation between the rows and the columns.

Chapter 6 discussed some of the open doors in this field of research. The QPBO algorithms is an interesting next-step for solving the MSS problem (and also the MWSDS, as the column generation scheme uses an MSS solver underneath). While the QPBO is an incomplete algorithm, an adaption of the algorithm and its inclusion in a complete solver could provide large speedup compared to the work presented in this thesis. Another interesting direction is composability: while efficient, the techniques presented in this thesis are ad-hoc and not composable: adding a new constraint to an existing problem is complex and will probably break existing filtering algorithms.

Bibliography

[Agr+96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. “Fast discovery of association rules”. In: *Advances in knowledge discovery and data mining*. USA: American Association for Artificial Intelligence, Feb. 1996, pp. 307–328. isbn: 978-0-262-56097-9. (Visited on 03/24/2020).

[AH14] C. C. Aggarwal and J. Han, eds. *Frequent Pattern Mining*. Springer International Publishing, 2014. doi: 10.1007/978-3-319-07821-2.

[All+16] D. Allouche, C. Bessiere, P. Boizumault, S. de Givry, P. Gutierrez, J. H. Lee, K. L. Leung, S. Loudni, J.-P. Métivier, T. Schiex, and Y. Wu. “Tractability-preserving transformations of global cost functions”. In: *Artificial Intelligence* 238 (2016), pp. 166–189. issn: 0004-3702. doi: https://doi.org/10.1016/j.artint.2016.06.005.

[AS+94] R. Agrawal, R. Srikant, et al. “Fast algorithms for mining association rules”. In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. Citeseer. 1994, pp. 487–499.

[BBV04] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BCR05] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. “Global constraint catalog”. In: (2005).

[BDT16] A. Backurs, N. Dikkala, and C. Tzamos. “Tight Hardness Results for Maximum Weight Rectangles”. In: *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Ed. by I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 81:1–81:13. isbn: 978-3-95977-013-2. doi: 10.4230/LIPIcs.ICALP.2016.81.

[Ben+02] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. "Discovering local structure in gene expression data: the order-preserving submatrix problem". In: *Proceedings of the sixth annual international conference on Computational biology*. 2002, pp. 49–57.

[BGN14] B. Babaki, T. Guns, and S. Nijssen. "Constrained Clustering Using Column Generation". In: *Integration of AI and OR Techniques in Constraint Programming*. Ed. by H. Simonis. Cham: Springer International Publishing, 2014, pp. 438–454. isbn: 978-3-319-07046-9.

[BH02] E. Boros and P. L. Hammer. "Pseudo-Boolean optimization". In: *Discrete Applied Mathematics* 123.1 (2002), pp. 155–225. issn: 0166-218X. doi: `https://doi.org/10.1016/S0166-218X(01)00341-9`.

[Bra+19a] V. Branders, G. Derval, P. Schaus, and P. Dupont. "Mining a Maximum Weighted Set of Disjoint Submatrices". In: *Discovery Science*. Ed. by P. Kralj Novak, T. muc, and S. Deroski. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 18–28. isbn: 978-3-030-33778-0. doi: `10.1007/978-3-030-33778-0_2`.

[Bra21] V. Branders. "Finding submatrices of maximal sum : applications to the analysis of gene expression data". PhD thesis. UCL - Université Catholique de Louvain, 2021. (Visited on 09/09/2021).

[BSD17a] V. Branders, P. Schaus, and P. Dupont. "Combinatorial Optimization Algorithms to Mine a Sub-Matrix of Maximal Sum". In: *International Workshop on New Frontiers in Mining Complex Patterns*. Springer. 2017, pp. 65–79.

[BSD17b] V. Branders, P. Schaus, and P. Dupont. "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*. 2017.

[BSD19] V. Branders, P. Schaus, and P. Dupont. "Identifying gene-specific subgroups: an alternative to biclustering". In: *BMC Bioinformatics* 20.1 (Dec. 2019), p. 625. issn: 1471-2105. doi: `10 . 1186 / s12859 - 019 - 3289 - 0`. (Visited on 03/24/2020).

[CC00] Y. Cheng and G. M. Church. "Biclustering of expression data." In: *Ismb*. Vol. 8. 2000. 2000, pp. 93–103.

[CF15] H. Cambazard and J.-G. Fages. “New filtering for AtMost-NValue and its weighted variant: A Lagrangian approach”. In: *Constraints* 20.3 (July 2015), pp. 362–380. issn: 1383-7133, 1572-9354. doi: `10.1007/s10601-015-9191-0`. (Visited on 03/05/2020).

[Coo+10] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. “Soft arc consistency revisited”. In: *Artificial Intelligence* 174.7 (2010), pp. 449–478. issn: 0004-3702. doi: `https://doi.org/10.1016/j.artint.2010.02.001`.

[CST+00] A. Califano, G. Stolovitzky, Y. Tu, et al. “Analysis of gene expression microarrays for phenotype classification.” In: *Ismb*. Vol. 8. 2000, pp. 75–85.

[Dao+18a] T. Dao, F. Docquier, M. Maurel, and P. Schaus. “Global migration in the 20th and 21st centuries: the unstoppable force of demography”. In: (2018).

[Dao+18b] T. H. Dao, F. Docquier, M. Maurel, and P. Schaus. *Global Migration in the 20th and 21st Centuries: the Unstoppable Force of Demography*. FERDI Working paper P223. Mar. 2018.

[Dao+21] T. H. Dao, F. Docquier, M. Maurel, and P. Schaus. “Global migration in the twentieth and twenty-first centuries: the unstoppable force of demography”. In: *Review of World Economics* 157.2 (May 2021), pp. 417–449. issn: 1610-2886. doi: `10.1007/s10290-020-00402-1`.

[De +97] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. “Computational geometry”. In: *Computational geometry*. Springer, 1997, pp. 1–17.

[Der+19b] G. Derval, V. Branders, P. Dupont, and P. Schaus. “The maximum weighted submatrix coverage problem: A CP approach”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by W.-J. van Hoeve. Springer International Publishing, 2019.

[Dhi01] I. S. Dhillon. “Co-Clustering Documents and Words Using Bipartite Spectral Graph Partitioning”. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’01. San Francisco, California: Association for Computing Machinery, 2001, pp. 269–274. isbn: 158113391X. doi: `10.1145/502512.502550`.

[DS20a] G. Derval and P. Schaus. *Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach: source code*. Version 0.1. Aug. 2020. doi: 10.5281/zenodo.3992317.

[DS20b] G. Derval and P. Schaus. *Software Open Access Maximal-Sum Submatrix search using a hybrid Contraint Programming/Linear Programming approach: experiment code and results*. Version 1.0. Zenodo, Aug. 2020. doi: 10.5281/zenodo.3992324.

[FLM99] F. Focacci, A. Lodi, and M. Milano. "Cost-based domain filtering". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 1999, pp. 189–203.

[Gay+15] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus. "Conflict Ordering Search for Scheduling Problems". In: *Principles and Practice of Constraint Programming*. Ed. by G. Pesant. Cham: Springer International Publishing, 2015, pp. 140–148. isbn: 978-3-319-23219-5.

[GG08] N. Gillis and F. Glineur. *Nonnegative Factorization and The Maximum Edge Biclique Problem*. 2008. arXiv: 0810.4225 [math.NA].

[GG13] N. Gillis and F. Glineur. "A continuous characterization of the maximum-edge biclique problem". In: *Journal of Global Optimization* 58.3 (Mar. 2013), pp. 439–464. doi: 10.1007/s10898-013-0053-2.

[GGM04] F. Geerts, B. Goethals, and T. Mielikäinen. "Tiling Databases". In: *Discovery Science*. Ed. by E. Suzuki and S. Arikawa. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 278–289. isbn: 978-3-540-30214-8. doi: 10.1007/978-3-540-30214-8_22.

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. isbn: 0716710455.

[GLD00] G. Getz, E. Levine, and E. Domany. "Coupled two-way clustering analysis of gene microarray data". In: *Proceedings of the National Academy of Sciences* 97.22 (2000), pp. 12079–12084.

[GNS09] I. Griva, S. G. Nash, and A. Sofer. *Linear and nonlinear optimization*. Vol. 108. Siam, 2009.

[Gui+18] D. Guillaume, B. Vincent, D. Pierre, and S. Pierre. *Synthetic dataset used in "The maximum weighted submatrix coverage problem: A CP approach"*. Nov. 2018. doi: 10.5281/zenodo.1688740.

[Gur18] L. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2018.

[Han+04] J. Han, J. Pei, Y. Yin, and R. Mao. "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach". In: *Data Mining and Knowledge Discovery* 8.1 (Jan. 2004), pp. 53–87. issn: 1573-756X. doi: 10.1023/B:DAMI.0000005258.31418.83. (Visited on 03/24/2020).

[Han+20] B. A. Han, S. M. ORegan, J. Paul Schmidt, and J. M. Drake. "Integrating data mining and transmission theory in the ecology of infectious diseases". In: *Ecology Letters* 23.8 (2020), pp. 1178–1188. doi: https://doi.org/10.1111/ele.13520. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ele.13520.

[Har72a] J. A. Hartigan. "Direct Clustering of a Data Matrix". In: *Journal of the American Statistical Association* 67.337 (1972), pp. 123–129. issn: 01621459.

[Har72b] J. A. Hartigan. "Direct Clustering of a Data Matrix". In: *Journal of the American Statistical Association* 67.337 (Mar. 1972), pp. 123–129. issn: 0162-1459. doi: 10.1080/01621459.1972.10481214. (Visited on 02/25/2019).

[HDT92] P. V. Hentenryck, Y. Deville, and C. Teng. "A Generic Arc-Consistency Algorithm and its Specializations". In: *Artif. Intell.* 57 (1992), pp. 291–321.

[HE80] R. M. Haralick and G. L. Elliott. "Increasing tree search efficiency for constraint satisfaction problems". In: *Artificial intelligence* 14.3 (1980), pp. 263–313.

[HHS84] P. L. Hammer, P. Hansen, and B. Simeone. "Roof duality, complementation and persistency in quadratic 01 optimization". In: *Mathematical Programming* 28.2 (Feb. 1984), pp. 121–155. issn: 1436-4646. doi: 10.1007/BF02612354. (Visited on 08/12/2021).

[Hur+16] B. Hurley, B. Osullivan, D. Allouche, G. Katsirelos, T. Schiex, M. Zytnicki, and S. De Givry. "Multi-language evaluation of exact solvers in graphical model discrete optimization". In: *Constraints* 21.3 (2016), pp. 413–434.

[IOC] IOC Research and Reference Service, The Guardian. *Olympic Sports and Medals, 1896-2014*. `https://www.kaggle.com/the-guardian/olympic-games`.

[Kar72] R. M. Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[KR07] V. Kolmogorov and C. Rother. "Minimizing nonsubmodular functions with graph cuts-a review". In: *IEEE transactions on pattern analysis and machine intelligence* 29.7 (2007), pp. 1274–1279.

[Krz+09] M. I. Krzywinski, J. E. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra. "Circos: An information aesthetic for comparative genomics". In: *Genome Research* (2009). doi: `10.1101/gr.092759.109`. eprint: `http://genome.cshlp.org/content/early/2009/06/15/gr.092759.109.full.pdf+html`.

[Le +14] T. Le Van, M. van Leeuwen, S. Nijssen, A. C. Fierro, K. Marchal, and L. De Raedt. "Ranked Tiling". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by T. Calders, F. Esposito, E. Hüllermeier, and R. Meo. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 98–113. isbn: 978-3-662-44851-9. doi: `10.1007/978-3-662-44851-9_7`.

[Ley09] M. Ley. "DBLP - Some Lessons Learned". In: *PVLDB* 2.2 (2009), pp. 1493–1500. doi: `10.14778/1687553.1687577`.

[LZ05] N. Lavra and B. Zupan. "Data Mining in Medicine". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by O. Maimon and L. Rokach. Boston, MA: Springer US, 2005, pp. 1107–1137. isbn: 978-0-387-25465-4. doi: `10.1007/0-387-25465-X_52`.

[Mac81] A. K. Mackworth. "Consistency in Networks of Relations". In: *Readings in Artificial Intelligence*. Ed. by B. L. Webber and N. J. Nilsson. Morgan Kaufmann, 1981, pp. 69–78. isbn: 978-0-934613-03-3. doi: `https://doi.org/10.1016/B978-0-934613-03-3.50009-X`.

[Med+02] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. "Traffic matrix estimation: Existing techniques and new directions". In: *ACM SIGCOMM Computer Communication Review*. Vol. 32. ACM. 2002, pp. 161–174.

[MO04] S. C. Madeira and A. L. Oliveira. “Biclustering algorithms for biological data analysis: a survey”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 1.1 (2004), pp. 24–45.

[MSV18] L. Michel, P. Schaus, and P. Van Hentenryck. *MiniCP: A Lightweight Solver for Constraint Programming*. Available from `https://minicp.bitbucket.io`. 2018.

[Mur] K. G. Murty. *Linear programming*. Springer.

[Nes83] Y. Nesterov. “A method for solving the convex programming problem with convergence rate O(1/k2)”. In: *Proceedings of the USSR Academy of Sciences* 269 (1983), pp. 543–547.

[Osc12] OscaR Team. *OscaR: Scala in OR*. Available from `https://bitbucket.org/oscarlib/oscar`. 2012.

[Ped99] W. Pedrycz. “Linguistic Data Mining”. In: *Computing with Words in Information/Intelligent Systems 2: Applications*. Ed. by L. A. Zadeh and J. Kacprzyk. Heidelberg: Physica-Verlag HD, 1999, pp. 399–420. isbn: 978-3-7908-1872-7. doi: `10.1007/978-3-7908-1872-7_19`.

[Pee03] R. Peeters. “The maximum edge biclique problem is NP-complete”. In: *Discrete Applied Mathematics* 131.3 (Sept. 2003), pp. 651–654. issn: 0166-218X. doi: `10.1016/S0166-218X(03)00333-0`. (Visited on 02/25/2019).

[PGA15] B. Pontes, R. Giráldez, and J. S. Aguilar-Ruiz. “Biclustering on expression data: A review”. In: *Journal of Biomedical Informatics* 57 (Oct. 2015), pp. 163–180. issn: 1532-0464. doi: `10.1016/j.jbi.2015.06.028`. (Visited on 03/24/2020).

[PSK14] A. P. Punnen, P. Sripratak, and D. Karapetyan. *The bipartite unconstrained 0-1 quadratic programming problem: polynomially solvable cases*. 2014. arXiv: `1212.3736 [math.OC]`.

[Rot+07] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. “Optimizing binary MRFs via extended roof duality”. In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[RVW06] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[SAG17] P. Schaus, J. O. R. Aoga, and T. Guns. “CoverSize: A Global Constraint for Frequency-Based Itemset Mining”. In: *Principles and Practice of Constraint Programming*. Ed. by J. C. Beck. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 529–546. isbn: 978-3-319-66158-2. doi: `10.1007/978-3-319-66158-2_34`.

[Sai+13] V. L. C. de Saint-Marcq, P. Schaus, C. Solnon, and C. Lecoutre. “Sparse-Sets for Domain Implementation”. In: *CP workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*. 2013, pp. 1–10.

[Sav97] M. Savelsbergh. “A Branch-and-Price Algorithm for the Generalized Assignment Problem”. In: *Operations Research* 45.6 (1997), pp. 831–841. doi: `10.1287/opre.45.6.831`.

[Sch99] C. Schulte. “Comparing Trailing and Copying for Constraint Programming.” In: *ICLP*. Vol. 99. 1999, pp. 275–289.

[Sel04] M. Sellmann. “Theoretical Foundations of CP-Based Lagrangian Relaxation”. In: *Principles and Practice of Constraint Programming CP 2004*. Ed. by M. Wallace. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 634–647. isbn: 978-3-540-30201-8. doi: `10.1007/978-3-540-30201-8_46`.

[Sha98] P. Shaw. “Using constraint programming and local search methods to solve vehicle routing problems”. In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.

[SMD03] Q. Sheng, Y. Moreau, and B. De Moor. “Biclustering microarray data by Gibbs sampling”. In: *Bioinformatics* 19.suppl_2 (2003), pp. ii196–ii205.

[Sou+08] M. C. de Souto, I. G. Costa, D. S. de Araujo, T. B. Ludermir, and A. Schliep. “Clustering cancer gene expression data: a comparative study”. In: *BMC bioinformatics* 9.1 (2008), p. 497.

[Tan08] J. Tan. “Inapproximability of Maximum Weighted Edge Biclique and Its Applications”. In: *Theory and Applications of Models of Computation*. Ed. by M. Agrawal, D. Du, Z. Duan, and A. Li. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 282–293. isbn: 978-3-540-79228-4. doi: `10.1007/978-3-540-79228-4_25`.

[The18] The World Bank. *Migration and Remittances Data*. data retrieved from "Bilateral Migration Matrix 2018", `http://www.worldbank.org/en/topic/migrationremittancesdiasporaissues/brief/migration-remittances-data`. 2018.

[TSS02] A. Tanay, R. Sharan, and R. Shamir. "Discovering statistically significant biclusters in gene expression data". In: *Bioinformatics* 18.Suppl 1 (July 2002), S136–S144. issn: 1367-4803, 1460-2059. doi: `10.1093/bioinformatics/18.suppl_1.S136`. (Visited on 11/22/2019).

[TZR18] M. C. Thomas, W. Zhu, and J. A. Romagnoli. "Data mining and clustering in chemical process databases for monitoring and knowledge discovery". In: *Journal of Process Control* 67 (2018). Big Data: Data Science for Process Control and Operations, pp. 160–175. issn: 0959-1524. doi: `https://doi.org/10.1016/j.jprocont.2017.02.006`.

[UF98] L. H. Ungar and D. P. Foster. "A Formal Statistical Approach to Collaborative Filtering". In: *In CONALD98*. 1998.

[Van+02] L. J. Van't Veer, H. Dai, M. J. Van De Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. Van Der Kooy, M. J. Marton, A. T. Witteveen, et al. "Gene expression profiling predicts clinical outcome of breast cancer". In: *nature* 415.6871 (2002), p. 530.

[Wol20a] L. Wolsey. "Branch and Bound". In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 7, pp. 113–138. isbn: 978-1-1196-0647-5. doi: `https://doi.org/10.1002/9781119606475.ch7`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch7`.

[Wol20b] L. Wolsey. "Column (and Row) Generation Algorithms". In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 11, pp. 213–233. isbn: 978-1-1196-0647-5. doi: `https://doi.org/10.1002/9781119606475.ch11`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch11`.

[Wol20c] L. Wolsey. "Complexity and Problem Reductions". In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 6, pp. 95–111. isbn: 978-1-1196-0647-5. doi: `https://doi.org/10.1002/9781119606475.ch6`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch6`.

[Wol20d] L. Wolsey. “Cutting Plane Algorithms”. In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 8, pp. 139–166. isbn: 978-1-1196-0647-5. doi: https://doi.org/10.1002/9781119606475.ch8. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch8.

[Wol20e] L. Wolsey. “Formulations”. In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 1, pp. 1–23. isbn: 978-1-1196-0647-5. doi: https://doi.org/10.1002/9781119606475.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch1.

[Wol20f] L. Wolsey. “Lagrangian Duality”. In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 10, pp. 195–212. isbn: 978-1-1196-0647-5. doi: https://doi.org/10.1002/9781119606475.ch10. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119606475.ch10.

[Xie+18] J. Xie, A. Ma, A. Fennell, Q. Ma, and J. Zhao. “It is time to apply biclustering: a comprehensive review of biclustering applications in biological and biomedical data”. In: *Briefings in Bioinformatics* 20.4 (Feb. 2018), pp. 1450–1465. issn: 1467-5463. doi: 10.1093/bib/bby014. (Visited on 03/24/2020).