Combinatorial Optimization Algorithms to Mine a Sub-Matrix of Maximal Sum

Vincent Branders(✉), Pierre Schaus, and Pierre Dupont

ICTEAM/INGI, Machine Learning Group,
Université catholique de Louvain,
Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium
{vincent.branders,pierre.schaus,pierre.dupont}@uclouvain.be

Abstract. Biclustering techniques have been widely used to identify homogeneous subgroups within large data matrices, such as subsets of genes similarly expressed across subsets of patients. Mining a *max-sum sub-matrix* is a related but distinct problem for which one looks for a (non-necessarily contiguous) rectangular sub-matrix with a maximal sum of its entries. Le Van et al. [7] already illustrated its applicability to gene expression analysis and addressed it with a constraint programming (CP) approach combined with large neighborhood search (LNS). In this work, we exhibit some key properties of this $\mathcal{NP}$-hard problem and define a bounding function such that larger problems can be solved in reasonable time. The use of these properties results in an improved CP-LNS implementation evaluated here. Two additional algorithms are also proposed in order to exploit the highlighted characteristics of the problem: a CP approach with a global constraint (CPGC) and a mixed integer linear programming (MILP). Practical experiments conducted both on synthetic and real gene expression data exhibit the characteristics of these approaches and their relative benefits over the CP-LNS method. Overall, the CPGC approach tends to be the fastest to produce a good solution. Yet, the MILP formulation is arguably the easiest to formulate and can also be competitive.

1 Introduction

Gene expression data is typically represented as a large matrix of gene expression levels across various samples. The study of such data is a valuable tool to improve the understanding of the underlying biological processes. For example, biomarker discovery aims at finding indicators of a disease or the physiological state of patients. This problem can be addressed with clustering techniques which perform a grouping of one dimension, either the rows or the columns of the original matrix. Yet it is known that breast cancer, for example, exhibits distinct subtypes [12,13]. In other words, specific genes exhibit activation patterns only in a specific group of patients. Biclustering techniques, or co-clustering, identify specific subsets of rows and of columns which jointly form homogeneous entries [9].

A. Appice et al. (Eds.): NFMCP 2017, LNAI 10785, pp. 65–79, 2018.
https://doi.org/10.1007/978-3-319-78680-3_5

In the present work, we focus on a related but different mining task. One looks in particular for subsets of rows and of columns with globally high values. In the context of gene expression analysis, the objective is to find a subset of genes which are relatively highly expressed among a subset of patients, even though some entries might depart from this pattern. With several thousands genes and hundreds of patients, we are mostly interested in the production of an algorithm that can deal with large matrices. Formally, one looks for a rectangular, and non necessarily contiguous, sub-matrix of a large matrix with a maximal sum of the selected entries. An illustrative example is provided in Fig. 1. The sum of the sub-matrix defined by the subset of rows $\{i_1, i_2, i_4, i_5\}$ and the subset of columns $\{j_2, j_4, j_5, j_6\}$ is 27.3 and is maximum. It can not be increased by the addition or the exclusion of any other row or column.

This *max-sum sub-matrix* problem is closely related to the *maximal ranked tile* mining problem studied by Le Van et al. [7]. In the later case, prior to the search of a sub-matrix, each matrix entry is replaced by its rank across its particular row. In other words, the maximal ranked tile mining is equivalent to the max-sum sub-matrix for which the matrix entries are discrete ranks along the rows. Since the combinatorial optimization algorithms to solve such problems are actually not specific to discrete entries we address here the slightly more general setting of continuous entries.

Our main contributions are (1) the study of the *max-sum sub-matrix* problem while exhibiting some of its key properties and the definition of an upper bound easy to compute in order to speed up the search for a solution; (2) the implementation of two additional algorithms making use of these properties: a CP approach with a global constraint (CPGC) and mixed integer linear programming (MILP); (3) practical experiments conducted both on synthetic and real gene expression data showing that the CP-LNS method can be largely outperformed; (4) the study of the relative benefits of the proposed methods across various problem instances.

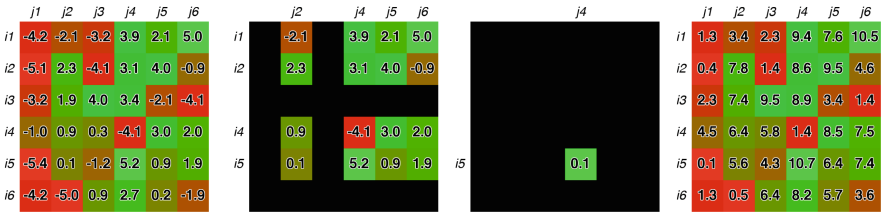
2 Problem

2.1 Statement

Definition 1 (The Max-Sum Sub-Matrix Problem). *Given a matrix* $\mathcal{M} \in \mathbb{R}^{m \times n}$ *consisting of* m *rows and* n *columns, let* $\mathcal{R} = \{1, \ldots, m\}$ *and* $\mathcal{C} = \{1, \ldots, n\}$ *be index sets for rows and for columns respectively, find the max-sum sub-matrix* (I^*, J^*)*, with* $I^* \subseteq \mathcal{R}$ *and* $J^* \subseteq \mathcal{C}$*, such that:*

$$(I^*, J^*) = \underset{I \subseteq \mathcal{R}, J \subseteq \mathcal{C}}{\operatorname{argmax}} f(I, J) = \underset{I \subseteq \mathcal{R}, J \subseteq \mathcal{C}}{\operatorname{argmax}} \sum_{i \in I, j \in J} \mathcal{M}_{i,j}. \tag{1}$$

The *objective function* $f(I, J)$ is the sum of the entries of a sub-matrix (I, J). The maximization term rewards, respectively penalizes, matrix entries with positive, respectively negative, values. One only considers matrices with positive and negative entries. Otherwise the optimal solution is trivially identified.



(a) Instance matrix (b) Max-sum sub-matrix (c) Threshold: 5.1 (d) Threshold: −5.5

Fig. 1. (a): Instance matrix. (b): Associated sub-matrix of maximal sum. (c) and (d): Sub-matrix of maximal sum obtained for different thresholds applied on the instance matrix.

This formulation implicitly assumes that there is a threshold equal to zero to consider entries as potentially relevant or informative.

A different threshold can be considered by subtracting some constant to all matrix entries. This allows to control the size of the optimal sub-matrix. Figures 1c and d illustrates solutions for a threshold of 5.1 and −5.5, respectively.

The max-sum sub-matrix problem is $\mathcal{NP}$-*hard* from a simple reduction[1] of the Maximum edge Weight Biclique Problem (MWBP) [4].

2.2 Explicit Versus Implicit Search Space

For a defined subset of columns J, the objective function can be formulated as $f(I, J) = \sum_{i \in I} r_i$ with r_i being the contribution of the row i:

$$r_i = \sum_{j \in J} \mathcal{M}_{i,j}. \tag{2}$$

This is important as the actual search can be limited to one dimension through independent computation of the contributions along the other dimension. Indeed, for any of the two dimensions being fixed, optimization along the other dimension is straightforward since it amounts to select only the subset of entries whose contribution is positive. For a fixed subset of columns $J \subseteq \mathcal{C}$, the subset of rows $I_J^* \subseteq \mathcal{R}$ that maximizes the objective value is identified as:

$$I_J^* = \underset{I \subseteq \mathcal{R}}{\operatorname{argmax}} \sum_{i \in I, j \in J} \mathcal{M}_{i,j} = \{i \in \mathcal{R} \mid r_i \geq 0\}. \tag{3}$$

In the gene expression analysis context, with order(s) of magnitude more rows (the genes) than columns (the samples), one typically searches for a subset of columns and selects the associated optimal subset of rows.

[1] Essentially by considering the rows and columns of the matrix as the two sets of nodes of a bipartite graph.

The search space of the max-sum sub-matrix problem contains $2^{|\mathcal{R}|} \times 2^{|\mathcal{C}|}$ *feasible solutions* that are rectangular sub-matrices ($I \subseteq \mathcal{R}, J \subseteq \mathcal{C}$) of the original matrix. Thanks to the independent contribution along one dimension, the number of feasible solutions to be explicitly evaluated is thus reduced to $2^{|\mathcal{C}|} \times |\mathcal{R}|$.

2.3 Related Work

Biclustering techniques address the problem of finding sub-matrices that satisfy some definition of homogeneity since entries grouped together into biclusters typically have similar values. A comprehensive review can be found in [9]. There is no assumption of homogeneity in the max-sum sub-matrix problem but rather one looks for a rectangular sub-matrix with an overall maximal sum. The difference is illustrated in Fig. 1b where a highly negative entry -4.1 in (i_4, j_4) is selected. This results from the sums of the selected entries in i_4 and in j_4 which contribute positively to the maximal sum. In the biclustering context, any entry that differs from entries of a bicluster or from entries in its row or its column is not expected to be selected. Cohesive biclusters [14,15], with high average values, are built by aggregating entries that are higher than a certain threshold such that the average value of the bicluster is higher than a second threshold. Then all entries must be higher than the first threshold while in the max-sum sub-matrix problem there is no expected minimum value for an entry to be selected. Biclustering approaches commonly identify multiple biclusters. Many algorithms iteratively identify a single bicluster and subsequently mask it [3,9,16]. Yang et al. [20] proposed a global alternative to these local optimal decisions. Designing a dedicated approach to the identification of multiple sub-matrices of maximal sum is part of our future research.

The maximum (contiguous) subarray problem introduced in [2] identifies a subarray of maximal sum from an array. For a one-dimensional array, this problem can be solved in linear time by Kadane's algorithm [2]. Cubic and subcubic time complexity algorithms have been proposed in the two-dimensional case [2,18,19]. This problem is however simpler than the max-sum sub-matrix since the selected sub-matrix is constrained to be formed of contiguous rows and contiguous columns from the original matrix.

The maximum ranked tile mining problem has been introduced in [7]. As discussed in Sect. 1, this is a special case of the max-sum sub-matrix problem for which the matrix entries are discrete ranks, corresponding to a permutation of column indices on each row. While the discrete ranking is an important characteristic of the maximum ranked tile mining problem, the associated constraint programming solution does not require discrete entries nor benefit from it. In this work, we present improved and new optimization approaches to solve the problem with arbitrary continuous entries.

Subgroup discovery is a data mining technique which extracts classification rules with respect to a target variable [1,6]. It departs from the standard learning of a

classifier as the extracted rules are not necessarily intended to cover all possible instances. Besides, such technique focuses on the interpretability of the classification rules rather than the generalization capability to classify new instances. Mining a max-sum sub-matrix is somewhat similar to subgroup discovery but there is no supervision by a specific target class variable. The hotspot detection problem, which can be considered as a particular form of subgroup discovery, aims at identifying subgroups that are unexpected with respect to some baseline information [5]. In the max-sum sub-matrix problem, one does consider globally high values without any baseline distribution of the data.

3 Optimization Approaches

3.1 Boolean Decision Vectors

The max-sum sub-matrix problem can be modeled with two vectors of boolean decision variables: $T = (T_1, \ldots, T_m)$ for the rows and $U = (U_1, \ldots, U_n)$ for the columns with $T_i \in \{0, 1\}$ and $U_j \in \{0, 1\}$. A sub-matrix (I, J) is defined by $I = \{i \in \mathcal{R} \mid T_i = 1\}$ and $J = \{j \in \mathcal{C} \mid U_j = 1\}$. The problem consists in assigning a value to each variable of T and U. Let us denote by $U^1 = \{j \in \mathcal{C} \mid U_j = 1\}$ the selected columns, $U^0 = \{j \in \mathcal{C} \mid U_j = 0\}$ the unselected ones and by $U^? = \{j \in \mathcal{C} \mid U_j = \{0, 1\}\}$ the undecided ones.

3.2 Constraint Programming

Constraint Programming (CP) is a flexible programing paradigm that is capable of solving optimization problems. As a declarative approach, it only requires to model the problem and, by using existing solvers, let it search and find solutions. A model is defined as a constraint satisfaction problem $CSP = (V, D, C)$ where V is the set of variables, D is their respective domains, and C is a set of constraints defined over the variables. A feasible solution is an assignment of the variables to values of their domains such that all constraints are satisfied.

Constraints are exploited to iteratively reduce the domains of variables. Such *constraint propagation* reduces the number of variable assignments to consider. Once all unfeasible values are removed from the domains of variables, the *fixpoint* of the propagation is reached. Then the solver selects a variable $X \in V$ that is unbound and recursively calls the solver while assigning a value to this variable. Through exploration of a depth-first-search tree (DFS), the solver either reaches a solution or backtracks when the domain of variables becomes empty.

Efficient backtracking is achieved through trailing, a state management strategy that facilitates the restoration of the computation state to an earlier version, effectively *undoing* changes that were imposed since then. The *trail* exposes two methods: `pushState` and `popState` to respectively time-stamp the current state and restore it. Its implementation is captured in terms of two simple stacks. The first stack holds entries to undo, the second one holds the frame sizes stacked between two consecutive call to `pushState`. Trailing enables the design of reversible objects defined on the trail.

In the rest of this section, the original model proposed by Le Van et al. [7] (*CP-LNS*$_0$) is presented, as well as an improved version (*CP-LNS*). Then a new constraint programming formulation is proposed (*CPGC*).

In CP-LNS$_0$, the tree reaches a leaf (or feasible solution) as soon as all rows and columns variables are bound. Each row and each column is associated to a reified constraint that ensures its selection if the sum of the entries along the other selected dimensions (respectively columns or rows) is positive. The LNS strategy is implemented as follows: after a given number of backtracking, the constraints on half of the columns variables of the best solution found so far are removed and the search restarts in another region. LNS may improve the time to identify a good solution at the cost of losing the possibility to prove optimality since the search is no longer complete.

In CP-LNS, by virtue of the implicit search space property (see Sect. 2.2), the tree reaches a leaf as soon as all column variables are bound. The contribution of all rows are computed afterwards. The objective function is computed as the sum of positive rows contributions. Similarly to CP-LNS$_0$, after some backtracking, half of the constraints, here only on the columns variables, are relaxed and a different region of the search space is explored.

CP-LNS$_0$. The max-sum sub-matrix problem has been modeled in [7] as:

$$\text{maximize} \sum_{i \in \mathcal{R}, j \in \mathcal{C}} T_i \times U_j \times \mathcal{M}_{i,j}, \tag{4}$$

$$\forall i \in \mathcal{R} : T_i = 1 \Leftrightarrow \sum_{j \in \mathcal{C}} U_j \times \mathcal{M}_{i,j} \geq 0, \tag{5}$$

$$\forall j \in \mathcal{C} : U_j = 1 \Leftrightarrow \sum_{i \in \mathcal{R}} T_i \times \mathcal{M}_{i,j} \geq 0. \tag{6}$$

Expression (4) states the optimization problem. The set of redundant constraints (5) and (6) permits a stronger filtering during the search.

CP-LNS. We propose here an improved CP model obtaining the same filtering as the original one but resulting in a lighter propagation of the constraints:

$$\text{maximize} \sum_{i \in \mathcal{R}} T_i \times r_i, \tag{7}$$

$$\forall i \in \mathcal{R} : T_i = 1 \Leftrightarrow r_i \geq 0. \tag{8}$$

The objective is to maximize the sum of rows contributions which are formalized as $r_i = \sum_{j \in \mathcal{C}} U_j \times \mathcal{M}_{i,j}$. Each row with positive (respectively negative) contribution is constrained in (8) to be selected (respectively unselected).

This improved model avoids the computation of the quite-heavy reified sum constraints (6) and reduces the number of terms in the objective function. As each product between variables in the objective is translated into a small binary constraint, reducing their number from $|\mathcal{R}| \times |\mathcal{C}|$ to $|\mathcal{R}|$ makes a significant difference on the time spent to compute the fix-point in each node.

CP Global Constraint. We also propose to improve the model due to Le Van et al. [7] by designing a novel algorithm encapsulated inside a global constraint that captures the whole problem. The pseudo-code is given in Algorithm 1. The call to the bounding and filtering procedures has been made explicit. In practice, the lines 11 to 14 would be encapsulated in the global constraint triggered by the fix-point algorithm. The key ingredients of our approach are:

- A feasible solution at each node of the search tree (`evaluate()`).
- An efficient bounding procedure (`upperBound()`).
- An efficient procedure to filter the domains (`filter()`).

A feasible solution at each node of the search tree. CP usually updates its feasible solution and best so far lower bound in the leaf-node of the search tree, that is when every variable of the problem is bound. One can observe that for the max-sum sub-matrix problem, any partial assignment of the variables can be extended implicitly as a complete solution for which the unbound variables would be set to 0 (i.e. $U^?$ variables are considered unselected). There is thus no need to wait that every variable is bound to evaluate the solution and possibly update the best so far lower bound. The value of the objective function of the feasible solution is computed in the `evaluate()` method as the sum of the positive rows contributions of the partial solution (see (2) and (3) where $U^1 = J$): $f(I^*_{U^1}, U^1) = \sum_{i \in \mathcal{R}} \max(0, r_i)$.

An efficient bounding procedure. CP uses a branch and bound depth-first-search to avoid the exploration of proven suboptimal solutions. The branch and bound performances depend on the strength and efficiency of the procedure to compute the upper bound. We design simple yet efficient bounding procedure for the max-sum sub-matrix problem. Intuitively one computes an upper bound on the row contribution of each row and sums up all the positive bounds on the rows. The upper bound on the contribution of a row is the sum of the matrix entries in the selected columns plus the sum of the positive entries in the unbound columns. One simply computes the upper bound as:

$$g(P) = g(U^1, U^0, U^?) = \sum_{i \in \mathcal{R}} \max(0, r_i + \sum_{j \in U^?} \max(0, \mathcal{M}_{i,j})). \tag{9}$$

The bound is admissible but not tight as it may optimistically evaluate the objective ($g(P) \geq f(P)$). Indeed, it relies on a per-line relaxation of the problem, each selecting a possibly different set of columns.

The `upperBound()` method is an implementation of the proposed upper bound using reversible double to store the incremental modifications of the partial contribution of the rows. Using a reversible sparse-set $\mathcal{T}$ for the row variables allows an efficient exclusion or inclusion of the rows through descent or backtrack [17]. Indeed, as soon as the bound on the row contribution becomes negative it should not be considered in the descendant nodes of the search tree. The number of rows to consider goes from exactly $|\mathcal{R}|$ to at most $|\mathcal{R}|$.

An efficient filtering procedure. The `filter()` method evaluates the upper bound result for two one-step look-ahead scenarios: if column j would be selected, this look-ahead upper bound is denoted $ub_j^{\in}$, or if j would not be selected, denoted $ub_j^{\notin}$. Then, any column j with $ub_j^{\in} \leq best$ is discarded as inclusion of the column can only lead to worst solution than the best so far. Similarly, any column j with $ub_j^{\notin} \leq best$ is included. The time complexity for computing all the look-ahead upper bounds is in $O(|\mathcal{T}| \times |U^?|)$. Indeed the look-ahead bound of each line for each column can be obtained in $O(1)$ from r_i^{ub}.

Algorithm 1. CP Global Constraint

```
1  best ← −∞  // best so far objective
2  trail
3  r: array[m] rev-double  // rows partial sums
4  T: rev-set ← {1, ..., n}  // candidate rows
5  Method dfs()
6      if U? ≠ φ then
7          j ← select j ∈ U?
8          foreach v ∈ {0, 1} do
9              trail.pushState()
10             D(Uj) ← v
11             best ← max(best, evaluate())
12             ub ← upperBound()
13             if ub > best then
14                 filter()
15                 dfs()
16             trail.popState()
17 Method evaluate(): double
       // evaluate objective
18     for j ∈ U? do
19         if D(Uj) ≠ {0, 1} then
20             U? ← U? \ j
21             if D(Uj) = 1 then
22                 ri ← ri + Mi,j , ∀i ∈ T
23     return Σ_{i∈T} max(0, ri)
24 Method upperBound(): double
25     ub ← 0
26     for i ∈ T do
27         r_i^ub ← ri + Σ_{j∈U?} max(0, Mi,j)
28         if r_i^ub > 0 then ub ← ub + r_i^ub
29         else T ← T \ i        // ri always ≤ 0
30     return ub
31 Method filter(): double
       // remove impossible values
32     ub_j^∈ ← 0 ∀j ∈ U?
33     ub_j^∉ ← 0 ∀j ∈ U?
34     for i ∈ T do
35         for j ∈ U? do
36             if Mi,j > 0 then
37                 if r_i^ub − Mi,j > 0 then
38                     ub_j^∉ ← ub_j^∉ + r_i^ub − Mi,j
39                 ub_j^∈ ← ub_j^∈ + r_i^ub
40             else
41                 ub_j^∉ ← ub_j^∉ + r_i^ub
42                 if r_i^ub + Mi,j > 0 then
43                     ub_j^∈ ← ub_j^∈ + r_i^ub + Mi,j
44     for j ∈ U? do
45         if ub_j^∈ ≤ best then D(Uj) ← 0
46         if ub_j^∉ ≤ best then D(Uj) ← 1
```

3.3 Mixed Integer Linear Programming

Mixed Integer Linear Programming [10] involves the optimization of a linear objective function, subject to linear constraints. Some or all of the variables are required to be integer. A MILP solver explores a branch and bound tree using linear-programming (LP) bounds at each node of the search tree.

It differs from a classical branch and bound as a LP *relaxation*, obtained by removing all the integrality constraints of a node, is solved before branching. The domain of all rows and columns variables changes from $\{0, 1\}$ to $[0, 1]$. This relaxed problem can be solved in polynomial time and the solution is an upper

bound on the objective value of the constrained problem. As an upper bound, the LP relaxation solution can be used to prune out suboptimal solutions.

If any integer variable is associated to a fractional value in the LP relaxation, two sub-problems are generated imposing restrictions on the domain of this variable. When all integrality constraints are satisfied in the solution of a node, then it corresponds to a feasible solution and the lower bound is possibly updated.

In an initial formulation, each entry of the matrix is associated to a decision variable that takes the value 1 if and only if both rows and columns are selected. The objective function is computed as the sum of the selected matrix entries.

Initial Model. The max-sum sub-matrix problem can be linearized as:

$$\text{maximize} \sum_{i \in \mathcal{R}, j \in \mathcal{C}} \mathcal{M}_{i,j} \times x_{i,j}, \tag{10}$$

$$\text{s.t. } x_{i,j} \leq T_i, \quad \forall i \in \mathcal{R}, \forall j \in \mathcal{C}, \tag{11}$$

$$x_{i,j} \leq U_j, \quad \forall i \in \mathcal{R}, \forall j \in \mathcal{C}, \tag{12}$$

$$x_{i,j} \geq T_i + U_j - 1, \quad \forall i \in \mathcal{R}, \forall j \in \mathcal{C}. \tag{13}$$

A binary decision variable is associated to each row T_i, to each column U_j and to each matrix entry $x_{i,j}$. The objective function is computed as the sum of matrix entries whose decision variable is set to one. Equations (11) to (13) enforce that variable $x_{i,j} = 1$ if and only if $T_i = 1$ and $U_j = 1$. All these decisions variables are relaxed to the interval $[0, 1]$ in the MILP solver.

Improved Model. In our experiments, reported in Sect. 4, we consider an improved and more compact MILP formulation. It relies on (3) where a row is selected if its contribution is positive and unselected otherwise. For each row, a variable r_i^+ is defined as the contribution of row i if it is positive and 0 otherwise. The objective is to maximize the sum over these variables. This linear model also uses "big M"constants. More specifically, (16) and (17) linearize $r_i^+ = \max(0, r_i)$ with r_i being the sum of the selected entries of row i.

$$\text{maximize} \sum_{i \in \mathcal{R}} r_i^+, \tag{14}$$

$$\text{s.t. } r_i = \sum_{j \in \mathcal{C}} \mathcal{M}_{i,j} \times U_j, \quad \forall i \in \mathcal{R}, \tag{15}$$

$$r_i^+ \leq T_i \times M_+, \quad \forall i \in \mathcal{R}, \tag{16}$$

$$r_i^+ \leq r_i + (1 - T_i) \times M_-, \quad \forall i \in \mathcal{R}. \tag{17}$$

If $T_i = 1$, $r_i^+ \leq r_i + (1 - T_i) \times M_- \leq T_i \times M_+$, then $r_i^+ \leq r_i$. If $T_i = 0$, $r_i^+ \leq T_i \times M_+ \leq r_i + (1 - T_i) \times M_-$, then $r_i^+ \leq 0$. From the maximization (14), it appears that if $r_i \geq 0$, T_i must be bound to 1 and $r_i^+ = r_i$. Otherwise, T_i must be bound to 0 and $r_i^+ = 0$. This formulation is valid if and only if $M_+ \geq r_i$ and $M_- + r_i \geq 0$. To avoid rounding errors and ill conditioned matrices, the big

M constants can be replaced by $\sum_{j \in \mathcal{C}} \max(0, \mathcal{M}_{i,j})$ and $-\sum_{j \in \mathcal{C}} \min(0, \mathcal{M}_{i,j})$, respectively in (16) and (17).

4 Experiments

This section describes experiments conducted to assess the relative performances of three algorithms to solve the max-sum sub-matrix problem. CP-LNS denotes the improved version of the method CP-LNS$_0$ proposed by Le Van [7]. The other algorithms are original methods proposed in the present work: a constraint programming with a global constraint (CPGC) and a mixed integer programming (MILP) solution.

These algorithms are first compared on data matrices which are generated in a controlled setting. Experiments on the breast cancer gene expression data used in [7] are reported next. The main criterion to assess the performance of the various methods is the computing time to solve a particular problem instance. This is technically assessed through an any-time profile defined below.

All algorithms have been implemented in the Scala programming language (2.11.4). Each run is executed with a single thread on a MacBook Pro (OS version 10.10.5) laptop (Intel i7-2720 CPU @ 2.20–3.30 GHz, 4 GB RAM per run). Constraint programming implementations are based on the latest version of OscaR [11] and MILP is based on the latest version of Gurobi (7.0.2). The code, datasets and supplementary results are available at https://bitbucket.org/vbranders/maxsumsubmatriximplementation.

4.1 Synthetic Data

We follow a similar protocol as in [7]. Synthetic data are generated by implanting a sub-matrix (I, J) of interest in a larger matrix $\mathcal{M} = (\mathcal{R}, \mathcal{C})$ made of m rows and n columns. The implanted sub-matrix (I, J) forms a specific selection of rows and columns chosen at random. For each row index (from 1 to m) and each column index (from 1 to n) of $\mathcal{M}$, a binary variable is sampled from a Bernoulli distribution $\mathrm{B}(p)$ and the associated row or column is included in the sub-matrix (I, J) if $\mathrm{B}(p) = 1$. Hence, $I = \{i \in \mathcal{R} \mid \mathrm{B}(p) = 1\}$ and $J = \{j \in \mathcal{C} \mid \mathrm{B}(p) = 1\}$. Next, the full matrix $\mathcal{M}$ is generated according to two normal distributions, $\mathcal{N}(1, 1)$ whenever the particular entry belongs to the implanted sub-matrix, and $\mathcal{N}(-3, 1)$ otherwise.

Such a generation protocol favors the occurrence of higher values in the implanted sub-matrix and lower values elsewhere. Yet, given the standard deviations chosen equal to 1, both ranges of values may overlap. We note that, as in [7], the implanted sub-matrix is not guaranteed to be an optimal solution to the max-sum sub-matrix problem. This generation protocol looks however realistic to define a rectangular (and not necessarily contiguous) sub-matrix of interest in a larger matrix.

Problem instances are generated for various matrix sizes (m, n) and a varying parameter p. As p increases, the size of the implanted sub-matrix is expected to increase as well. In the gene expression analysis context, m can easily be two orders

of magnitude larger than n and the sub-matrix of interest is typically small as compared to the full matrix. Such cases are included in the controlled experiments reported below but a larger spectrum of problem instances is also considered.

4.2 Gene Expression Data

The proposed case study concerns biomarker discovery for breast cancer subtypes using heterogeneous molecular data types. For a biological analysis and interpretation of the results, the reader is redirected to the work of [7]. The pre-processed data provided by [7] consists of a matrix of $m = 2,211$ rows and $n = 94$ columns.

Matrix entries are first transformed to discrete ranks along each row. A given threshold $\theta \times n$ is then subtracted from each entry. As θ increases, the proportion of positive entries decreases and, consequently, a smaller sub-matrix of interest is expected to be found. Hence, the control parameter θ plays a similar role as the parameter p (from Sect. 4.1) but in an opposite way.

4.3 Evaluation

One could assess algorithms performances through runtime or number of feasible solutions. While the former may depend on implementation details, the latter strongly depends on the time spent in each node. As an example, the large number of reified constraints in CP-LNS$_0$ has a major impact on the time spent to compute the fix-point in each node while the filtering is as strong as the filtering of the CP-LNS model. While both should perform equally well in terms of the number of feasible solutions, it was observed in preliminary experiments that CP-LNS$_0$ is significantly slower than CP-LNS (the interested reader may consult https://bitbucket.org/vbranders/maxsumsubmatriximplementation). We prefer the runtime comparisons as it is a more common approach and we made sure to implement the algorithms in the most comparable fashion.

Any-Time Profile. In practice, an important criterion for the user is the time required to solve an instance and the ability to find the best solution within a given budget of time. Using *any-time profiles*, one can summarize these characteristics. The idea behind any-time profiles is that an algorithm should produce as high quality solution as possible at any moment of its running time [8]. It directly provides a cumulative probability for a method to solve an arbitrary instance after a given budget of time. In the max-sum sub-matrix problem, a high solution quality corresponds to a sub-matrix of large sum. For each instance, runs not completed in a maximum budget of time $t^{\max}$ are interrupted.

Definition 2 (Max-Sum Sub-Matrix Any-Time Profile). *Let* $f(algo, inst, t)$ *be the objective value of the best solution found so far by an algorithm* $algo$ *for an instance* $inst$ *at time* t. *Let* $t^{\max}$ *be the maximum running time before*

interrupting an algorithm. The any-time profile of an algorithm is the solution quality $Q_{algo}(t)$ *computed on all instances as a function of the time:*

$$Q_{algo}(t) = \frac{1}{|inst|} \sum_{inst} \frac{f(algo, inst, t)}{f(algo^*_{inst}, inst, t^{\max})}, \tag{18}$$

with $algo^*_{inst} = \underset{algo}{\operatorname{argmax}} f(algo, inst, t^{\max})$.

4.4 Results

Figure 2 presents the any-time profile on 50 synthetic data with $10,000$ rows and $p = \{0.05, 0.3, 0.7\}$ for 100 columns (column 1) or $1,000$ columns (column 2) and the any-time profile on breast cancer gene expression data with $2,211$ rows, 94 columns and variable choices of θ (columns 3 and 4).

Synthetic Data. The CP-LNS method is clearly outperformed by the two other methods. It can barely produce any solution within the allocated time budget. The best approach is CPGC followed by MILP. The reported curves are stopped whenever the proof of optimality is obtained or else the maximal running time is reached. Hence, CPGC also exhibits best results whenever proving optimality is possible in the allocated running time.

Gene Expression Data. Each curve corresponds here to the performance of an algorithm on a single instance, the one obtained for a specific choice of θ. On the whole spectrum of instances considered, the clear winner is CPGC. The most interesting instances are those for which $\theta \geq 0.9$ since such settings correspond to small sub-matrices which are more likely to illustrate an interesting biological pattern. In such cases, the best approaches are CPGC and CP-LNS.

Summary. As expected by the size of the search tree, CP-LNS is sensible to the size of the instance matrix producing barely no results on the larger synthetic instances within the time budget. On the opposite, CPGC achieves the best results. Indeed the model uses a dedicated global constraint with efficient filtering through computation of an upper bound and fast update of the lower bounds. The results of MILP are surprisingly good given its inability to express specialized constraints such as these of CP. This is explained by the benefits of the linear-programming relaxation to tighten the gap between the lower and the upper bounds. The current major issue is related to the "big M" approach that fails to guide the search in some settings on the gene expression data. When θ is smaller, the "big M" constant M_- is tighter. As a consequence, the result of the LP relaxation as a higher chance to be a tighter bound. It follows a speed-up of the search as it implies a more efficient pruning of the tree.

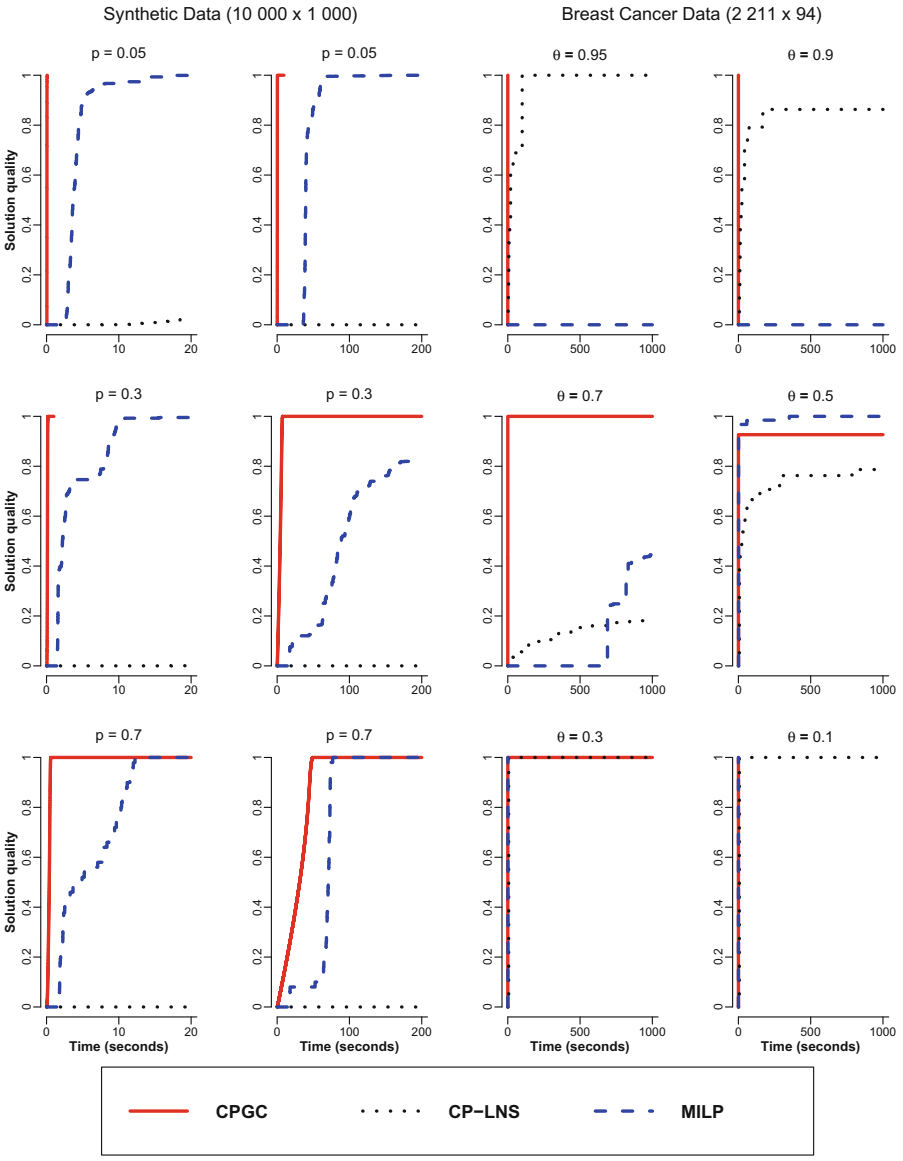


Fig. 2. Any-time profiles of constraint programming with a global constraint (CPGC), the CP method with large neighborhood search (CP-LNS) and mixed integer linear programming (MILP).
Columns 1 and 2: reported curves correspond to the average solution quality over all **synthetic instances** as a function of time (in seconds). Results are computed on 50 **synthetic instances** with 10,000 rows, 100 (column 1) or 1,000 (column 2) columns, a variable p and a maximum running time of 20 (column 1) or 200 (column 2) seconds. **Columns 3 and 4:** reported curves correspond to the solution quality over each **gene expression problem instance** obtained for a specific θ as a function of the time (in seconds). Results are computed on breast cancer **gene expression data** with 2,211 rows, 94 columns and various θ values for a maximum CPU time of 1,000 s.

5 Conclusions and Perspectives

We introduce the *max-sum sub-matrix* problem which consists in finding a (non necessarily contiguous) rectangular sub-matrix in a large matrix whose sum is maximal. This problem is originally motivated, in the context of gene expression analysis, by the search of a subset of highly expressed genes in a specific subset, to be found, of relevant samples exhibiting such a pattern. A close variant of this problem, known as *maximal ranked tile* mining problem, has already been studied and tackled with constrained programming (CP) combined with large neighborhood search (LNS) [7].

We present here key properties of the max-sum sub-matrix problem to speed up the search for a solution. This results in an improved CP-LNS implementation. We also propose two new algorithms to solve this problem. Experiments reported both on synthetic data and the original gene expression data used in [7] illustrate the benefits of our proposed methods. In particular, a CP approach with a global constraint (CPGC) is the most effective one in a large spectrum of problem instances. Overall, the CPGC method is also best at proving optimality when such proof can be obtained within the allocated CPU time budget.

The second approach proposed here relies on mixed integer linear programming (MILP). It is arguably the simplest to formulate and to address with a standard solver for such problems. It is competitive with the other methods and largely outperforms CP-LNS as well in our controlled experiments. It exhibits however some performance degradation on some instances from gene expression data, most likely as a consequence of the specific relaxation it is based on.

The max-sum sub-matrix mining problem could be extended to a supervised classification setting. For example, in gene expression analysis, one typically wants to find genes (rows) that allows to discriminate between two conditions. In other words, the columns could be *a priori* labeled according to two conditions. The objective can then be to identify a subset of rows that are maximally relevant to discriminate between subsets of samples from different conditions. This could be encoded in a larger matrix for which columns represent pairs of columns in either conditions from the original matrix and the value stored is interpreted as a distance value for a particular gene across both conditions.

The max-sum sub-matrix problem could also be applied to outlier detection and/or biclustering. For example, using an appropriate data transformation, entries that are close to the mean or to the median could be mapped to relatively large positive entries. Similarly, entries far away from the mean would be mapped to low values. Consequently a sub-matrix of maximal sum after such transformation would correspond to subsets of rows and of columns exhibiting similar entries. Explicit comparisons to existing biclustering algorithms could be considered in such a setting.

References

1. Atzmueller, M.: Subgroup discovery. Wiley Interdiscipl. Rev. Data Mining Knowl. Discov. **5**(1), 35–49 (2015)
2. Bentley, J.: Programming pearls: algorithm design techniques. Commun. ACM **27**(9), 865–873 (1984)
3. Cheng, Y., Church, G.M.: Biclustering of expression data. In: ISMB, vol. 8, pp. 93–103 (2000)
4. Dawande, M., Keskinocak, P., Tayur, S.: On the biclique problem in bipartite graphs (1996)
5. Fanaee-T, H., Gama, J.: Eigenspace method for spatiotemporal hotspot detection. Expert Syst. **32**(3), 454–464 (2015). eXSY-Nov-13-198.R1
6. Herrera, F., Carmona, C.J., González, P., del Jesus, M.J.: An overview on subgroup discovery: foundations and applications. Knowl. Inf. Syst. **29**(3), 495–525 (2011)
7. Le Van, T., van Leeuwen, M., Nijssen, S., Fierro, A.C., Marchal, K., De Raedt, L.: Ranked tiling. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) ECML PKDD 2014. LNCS (LNAI), vol. 8725, pp. 98–113. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44851-9_7
8. López-Ibánez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. Eur. J. Oper. Res. **235**(3), 569–582 (2014)
9. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) **1**(1), 24–45 (2004)
10. Nemhauser, G.L., Wolsey, L.A.: Integer programming and combinatorial optimization. Wiley, Chichester (1988). Nemhauser, G.L., Savelsbergh, M.W.P., Sigismondi, G.S.: Constraint classification for mixed integer programming formulations. COAL Bull. **20**, 8–12 (1992)
11. OscaR Team: OscaR: Scala in OR (2012). https://bitbucket.org/oscarlib/oscar
12. Parker, J.S., Mullins, M., Cheang, M.C., Leung, S., Voduc, D., Vickery, T., Davies, S., Fauron, C., He, X., Hu, Z., et al.: Supervised risk predictor of breast cancer based on intrinsic subtypes. J. Clin. Oncol. **27**(8), 1160–1167 (2009)
13. Perou, C.M., Sørlie, T., Eisen, M.B., van de Rijn, M., Jeffrey, S.S., Rees, C.A., Pollack, J.R., Ross, D.T., Johnsen, H., Akslen, L.A., et al.: Molecular portraits of human breast tumours. Nature **406**(6797), 747–752 (2000)
14. Pio, G., Ceci, M., D'Elia, D., Loglisci, C., Malerba, D.: A novel biclustering algorithm for the discovery of meaningful biological correlations between micrornas and their target genes. BMC Bioinform. **14**(7), S8 (2013)
15. Pio, G., Ceci, M., Malerba, D., D'Elia, D.: Comirnet: a web-based system for the analysis of mirna-gene regulatory networks. BMC Bioinform. **16**(9), S7 (2015)
16. Pontes, B., Giráldez, R., Aguilar-Ruiz, J.S.: Biclustering on expression data: a review. J. Biomed. Inform. **57**, 163–180 (2015)
17. de Saint-Marcq, V.l.C., Schaus, P., Solnon, C., Lecoutre, C.: Sparse-sets for domain implementation. In: CP Workshop on Techniques foR Implementing Constraint programming Systems (TRICS), pp. 1–10 (2013)
18. Takaoka, T.: Efficient algorithms for the maximum subarray problem by distance matrix multiplication. Electron. Not. Theoret. Comput. Sci. **61**, 191–200 (2002)
19. Tamaki, H., Tokuyama, T.: Algorithms for the maximum subarray problem based on matrix multiplication. In: SODA 1998, pp. 446–452 (1998)
20. Yang, J., Wang, H., Wang, W., Yu, P.: Enhanced biclustering on expression data. In: Proceedings of the Third IEEE Symposium on Bioinformatics and Bioengineering, pp. 321–327. IEEE (2003)