A Fixed Parameter Tractable Integer Program for Finding the Maximum Order Preserving Submatrix

Jens Humrich, Thomas Gaertner, Gemma Garriga

A Fixed Parameter Tractable Integer Program for Finding the Maximum Order Preserving Submatrix

Jens Humrich*, Thomas Gärtner*†, and Gemma C. Garriga‡
* Department of Computer Science, University of Bonn, Germany
† Knowledge Discovery Department, Fraunhofer IAIS, Germany
‡ Mostrare research team, INRIA Lille Nord Europe, France

***Abstract*—Order-preserving submatrices are an important tool for the analysis of gene expression data. As finding large order-preserving submatrices is a computationally hard problem, previous work has investigated both exact but exponential-time as well as polynomial-time but inexact algorithms for finding large order-preserving submatrices. In this paper, we propose a novel exact algorithm to find maximum order preserving submatrices which is fixed parameter tractable with respect to the number of columns of the provided gene expression data. In particular, our algorithm is based on solving a sequence of mixed integer linear programs and it exhibits better guarantees as well as better runtime performance as compared to the state-of-the-art exact algorithms. Our empirical study in benchmark datasets shows large improvement in terms of computational speed.**

I. Introduction

Since the introduction of the order-preserving submatrix problem for gene expression data by Ben-Dor et al in 2003 [1], a large variety of solutions has been suggested in literature. The motivations for the different solutions were the need for (i) exact solutions, (ii) theoretical guarantees, (iii) scalable algorithms, and (iv) solutions robust to noise. In this paper, we address all four desiderata by one algorithm.

Order preserving submatrices were proposed as interesting local patterns in gene expression data to achieve a higher robustness with respect to the actual expression levels [1]. An *order preserving submatrix* of a matrix $A \in \mathbb{R}^{n\times m}$ is a submatrix, induced by a set of rows $R \subseteq \{1,\dots,n\}$ and columns $C \subseteq \{1,\dots,m\}$, for which an ordering π of the columns exist, such that along this ordering the values of every row of the submatrix are strictly increasing. Each entry $A(i,j)$ corresponds to the readout associated to row (gene) i and column (experiment) j of the matrix A and typically it holds that $n \gg m$. Two pattern mining problems naturally arise in the context of order preserving submatrices: (a) enumerating all order preserving submatrices and (b) finding a largest order preserving submatrix. The enumeration problem suffers from the potentially exponential output and from having to select the interesting ones of those. The maximisation problem can be shown to be NP-hard by reducing it to its decision version, which is know to be NP-complete [1]. Therefore we will investigate algorithms for the maximisation problem with, in the worst case, exponential run time.

To find the largest order preserving submatrix it is sufficient to enumerate all possible subsets of rows and columns [2], as their permutation can be derived from the entries. This results in an algorithm with run time exponential in $n+m$. To avoid the brute-force enumeration in some cases, Trapp and Prokopyev [2] proposed an integer linear program. While empirically the MILP of Trapp and Prokopyev is in many cases much faster than brute-force enumeration, as we will discuss later, their algorithm has worse theoretical guarantees, its run time is exponential not only in $n+m$ but also in nm. In this paper, we propose a novel MILP which is fixed parameter tractable with parameter m, that is, that has run time exponential in m but only polynomial in n. Experimental results confirm the superior performance of our formulation. Building on this improvement, we also propose solutions to a noise-tolerant maximum order preserving submatrix problem.

II. Preliminaries

Consider a real-valued matrix $A \in \mathbb{R}^{n\times m}$. Let $[\![n]\!]$ denote the set $\{1,\dots,n\}$. For subsets of row and column indices $R \subseteq [\![n]\!], C \subseteq [\![m]\!]$, and a permutation $\pi : [\![m]\!] \leftrightarrow [\![m]\!]$ the submatrix $A(R,\pi(C))$ is the result of arranging the columns in C of the submatrix given by $R \times C$ in A according to π. A triple (R,C,π) of rows R, columns C, and permutation π induces an *order-preserving submatrix* in $A \in \mathbb{R}^{n\times m}$ if

$A(R, \pi(C))$ is ordered, that is, if for all $i \in R$ and all $j, j' \in C$ it holds that $\pi(j) < \pi(j')$ implies $A(i, j) < A(i, j')$. Such triples (R, C, π) will be called OPSM-patterns.

The central decision problem of OPSM is:

Problem 2.1 ($\exists$-OPSM*):* Given a matrix $A \in \mathbb{R}^{n \times m}$ and parameters $\rho \leq n, \gamma \leq m$, decide if there is an OPSM-pattern (R, C, π) with $|R| = \rho$ and $|C| = \gamma$.

It has been shown that $\exists$-OPSM is NP-complete [1].

In the optimization version of this problem, we are interested in finding the maximum OPSM-pattern with respect to $|R||C|$ for subsets of rows $R \subseteq [\![n]\!]$ and columns $C \subseteq [\![m]\!]$. As the decision problem is NP-complete, the associated optimization problem is NP-hard. It is formally defined as:

*Problem 2.2 (*MAX-OPSM*):* Given a matrix $A \in \mathbb{R}^{n \times m}$, find a OPSM-pattern (R, C, π) such that $|R||C|$ is maximal.

The MAX-OPSM problem was studied in [2], where the authors propose an integer programming based algorithm.

A. Tractability and Brute-Force Search

As mentioned in the introduction, to find the largest OPSM-pattern it is sufficient to enumerate all possible subsets of rows and columns [2], as their permutation can be derived from the entries. Using this observation, we have a run time for the brute force search of $O(n \exp(n + m))$ instead of the full $O(m! \exp(n + m))$.

As n can be very large, an algorithm with computational complexity larger than $\exp(n)$ is intractable. In contrast, as m is typically very small in the applications we consider, complexities of the form $\text{poly}(n) \exp(m)$ can still be considered tractable. Such algorithms are often called *fixed-parameter tractable*, in this case, with respect to the parameter m. To achieve such an algorithm, the following observation [1] is central: Given a set of columns C and a permutation π, the set of rows R such that (R, C, π) is an OPSM-pattern and such that $|R||C|$ is maximal, can be found in time linear in nm. Hence, it is easy to derive an improved brute-force algorithm with run time $O(n^2 m \exp(m))$.

Let now π_i denote the permutation that puts the i-th row in increasing order and by $\hat{A}_i = A([i{+}1 : n], \pi_i([\![m]\!])) \in \mathbb{R}^{\hat{n} \times m}$, the submatrix of A starting with the $i+1$-th row up to the last row, whose columns are permuted according to π_i. It is now sufficient to enumerate sets of columns and check for which rows these columns are increasing in each matrix of the set $\{\hat{A}_i\}_i$. Notice that the i-th row itself can be skipped, since it is in increasing order.

To analyse the time complexity of integer as well as mixed integer linear programs (MILP) we assume a MILP solver with run time exponential in the number of binary variables and cubic in the number of real variables. Such a MILP solver is easily attained by brute-force enumeration and a much better worst-case run time can not be hoped for.

B. Related work

Ben-Dor et al. [1] proposed originally a heuristic algorithm for mining OPSM patterns based on a stochastic model which adds rows to an OPSM pattern based on their fitness. The complete enumeration of all OPSM patterns is not ensured, neither having found the solution of the maximum OPSM among those patterns. Rho and Park [3], described a genetic algorithm which adds new columns depending on the support of rows for the particular order. Gao et al. [4] proposed an algorithm for finding OPSM patterns by a row growing strategy with a constrained solution space.

In [5], the authors propose to mine for relaxed OPSM-patterns, defined as patterns that are similar in terms of overlapping of the longest common subsequence, to a hidden backbone order in the data. Their algorithm does not ensure complete enumeration. In [6] the proposal is to mine for OPSM patterns where the attribute order might be slightly violated; patterns are then grouped in clusters that deviate from the concensus order by up to a certain fraction of attributes. None of these algorithms can make an exhaustive enumeration, nor they approach the optimization problem related to finding maximum tolerant OPSM.

The first paper to work on MAX-OPSM is Trapp and Prokopyev [2]. They motivate the need to have exact optimization approaches making use of mixed-integer programming in order to facilitate the theoretical analysis of the problem and practical interpretation of the result. Our approach builds on this work and we therefore describe their approach in more detail in the next section.

C. Exact MAX-OPSM by Integer Programming

We next consider the integer linear programming formulation proposed by Trapp and Prokopyev [2]. The main idea is to find the largest row and column subsets (R, C) for every matrix in $\{\hat{A}_r\}_r$ such that $\hat{A}_r(R, C)$ is ordered. Let $\hat{n}_r = n - r$ and as the index r will usually clear from the context we will from now use $\hat{A}, \hat{n}$ instead of $\hat{A}_r, \hat{n}_r$. The optimisation in each step is then over $\hat{n}$ binary variables x_i, m binary

variables y_i, and $\hat{n} \times m$ binary variables z_{ij}. We will refer to the integer program formulation of Trapp and Prokopyev [2] by CF-OPSM (for compact formulation of the order preserving submatrix). It is defined as:

$$\underset{X,Y,Z}{\operatorname{argmax}} \sum_i \sum_j z_{ij}$$

$$\text{s.t.} \begin{cases} z_{ij} + z_{ik} \leq x_i \text{ for all } j < k \text{ , and} \\ \qquad \hat{A}(i,j) \geq \hat{A}(i,k) \text{ for } i \in [\![\hat{n}]\!] & \text{(1a)} \\ z_{ij} \leq x_i \text{ for all } i \in [\![\hat{n}]\!], j \in [\![m]\!] & \text{(1b)} \\ z_{ij} \leq y_j \text{ for all } i \in [\![\hat{n}]\!], j \in [\![m]\!] & \text{(1c)} \\ z_{ij} \geq x_i + y_j - 1 \text{ for all } i \in [\![\hat{n}]\!], j \in [\![m]\!] & \text{(1d)} \\ X \in \{0,1\}^{\hat{n}}, Y \in \{0,1\}^m, Z \in \{0,1\}^{\hat{n} \times m} & \text{(1e)} \end{cases}$$

Constraints (1b) to (1d) enforce that $z_{ij} = x_i y_j$, that is, an entry is considered as 'selected' if and only if the corresponding row and column is 'selected'. Constraint (1a) ensures that the solution of CF-OPSM is such that $\hat{A}_r(C^*, R^*)$ with $R^* = \{i \mid x_i = 1\}$ and $C^* = \{j \mid y_j = 1\}$ is ordered. For that it prevents entries from being included in the solution that contradict the permutation of row r. This optimisation problem has $\hat{n} + m + \hat{n}m$ binary variables and at most $3\hat{n}m + \frac{1}{2}\hat{n}m(m-1)$ constraints. It is hence not fixed-parameter tractable with respect to the parameter m.

To overcome the severe run time demands, Trapp and Prokopyev [2] added further constraints in their integer program, that we will describe in the next subsection. Most importantly they added constraints on the solution space such that not always the largest OPSM-pattern could be found. In Section V we will compare our algorithm with the run time of the best implementation proposed by Trapp and Prokopyev on similar hardware.

III. Fixed Parameter Tractable Max-Opsm

In this section we will develop a novel mixed integer linear programming based approach for solving MAX-OPSM exactly. The main achievement of this algorithm is that it is fixed parameter tractable with respect to the number of columns of the given matrix as the parameter. Along the way, we will show how the number of variables as well as constraints can be reduced and how upper and lower bounds on the size of the solution can help speed up the overall algorithm.

A. Reducing the Number of Variables

Most of the constraints in Equation (1) serve only to ensure that $z_{ij} = x_i y_j$. If we can rewrite the objective function and constraint (1a) in terms of x_i and y_j only, we will thus be able to drop z_{ij}. For the target function this rewriting is trivial as $\sum_i \sum_j z_{ij}$ is equivalent to $\sum_i x_i \sum_j y_j$. To address the constraints, consider first the case $x_i = 0$: In this case constraint (1b) enforces both $z_{ij} = 0$ and $z_{ik} = 0$ and constraint (1a) is trivially ensured. In the case that $x_i = 1$, however, constraint (1a) is needed to ensure that the solution is a valid OPSM, i.e., that $y_j + y_k \leq 1$ for conflicting columns k, j. It can thus be replaced by the equivalent condition $y_j + y_k \leq 2 - x_i$, leading to:

$$\underset{X,Y}{\operatorname{argmax}} \sum_i x_i \sum_j y_j$$

$$\text{s.t.} \begin{cases} y_j + y_k \leq 2 - x_i \text{ for all } j < k, \text{ and} \\ \qquad \hat{A}(i,j) \geq \hat{A}(i,k) \text{ for } i \in [\![\hat{n}]\!] \\ X \in \{0,1\}^{\hat{n}}, Y \in \{0,1\}^m. \end{cases}$$

Observe that this problem has only $\hat{n} + m$ binary variables and at most $\frac{1}{2}\hat{n}m(m-1)$ constraints. On the other hand, the objective function is now non-linear. Indeed, even maximising the continuous relaxation of this quadratic target function is unlikely to be possible in polynomial time, as the objective function is non-convex. To solve this optimisation problem, we replace it by a sequence of problems for $\gamma = 2, \ldots, m$, each with the constraint $\sum_j y_j \geq \gamma$. We now have the following:

$$\underset{X,Y}{\operatorname{argmax}} \sum_i x_i$$

$$\text{s.t.} \begin{cases} y_j + y_k \leq 2 - x_i \text{ for all } j < k, \text{ and} \\ \qquad \hat{A}(i,j) \geq \hat{A}(i,k) \text{ for } i \in [\![\hat{n}]\!] \\ \sum_j y_j \geq \gamma \\ X \in \{0,1\}^{\hat{n}}, Y \in \{0,1\}^m \end{cases} \quad \text{(3a)}$$

This problem has only $\hat{n} + m$ binary variables, at most $1 + \frac{1}{2}\hat{n}m(m-1)$ constraints, and needs to be solved nm times.

B. Reducing the Number of Constraints

Trapp and Prokopyev [2] proposed to combine several inequality constraints belonging to the same row. As $n \gg m$, however, a better reduction is possible if we combine the constraints belonging to the same columns. Consider a set of rows $\{i_1, \ldots, i_l\}$ in $\hat{A}$ for which the same columns j and k with $j < k$ satisfy $\hat{A}(r,j) \geq A(r,k)$ for all $r \in \{i_1, \ldots, i_l\}$. Since all variables are binary, we can combine the constraints (3a) for such sets of rows into $y_j + y_k \leq 2 - \frac{1}{l}(x_{i_1} - x_{i_2} - \ldots x_{i_l})$.

This way we reduce all constraints for a pair of columns into one single constraint. To compute the constraints we loop over all possible pairs j, k of columns such that $j < k$ and check whether certain row entries are ascending. This can be done in $O(n^2 m)$. The reduced optimisation problem is:

$$\operatorname*{argmax}_{X,Y} \sum_i x_i$$

$$\text{s.t.} \begin{cases} y_j + y_k \leq 2 - \dfrac{1}{|I_{jk}|} \displaystyle\sum_{i \in I_{jk}} x_i \text{ for all } j < k & \text{(4a)} \\ \displaystyle\sum_j y_j = \gamma & \text{(4b)} \\ X \in \{0,1\}^{\hat{n}}, Y \in \{0,1\}^m & \text{(4c)} \end{cases}$$

with $I_{jk} = \{i : \hat{A}(i,j) \geq \hat{A}(i,k)\}$.

This linear program has only $m+n$ binary variables and at most $1 + \frac{1}{2}m(m-1)$ constraints.

C. Relaxing the Variables

To derive an optimisation problem that is fixed parameter tractable with respect to m, we relax the constraint $X \in \{0,1\}^{\hat{n}}$ to $X \in [0,1]^{\hat{n}}$ and observe the implication of constraint (4a) and the objective function on the solution vector. We distinguish two cases according to the value of the binary variables y_j and y_k.

First consider the case that $y_j + y_k = 2$, i.e., that $y_j = 1$ and $y_k = 1$. The constraint (4a) is then $\sum_{i \in I_{jk}} x_i \leq 0$. Since $X \in [0,1]^{\hat{n}}$ this implies $x_i = 0$ for all $i \in I_{jk}$. In the case that $y_j + y_k < 2$, one of the variables, y_j or y_k, is equal to zero and constraint (4a) becomes $\sum_{i \in I_{jk}} x_i \leq |I_{jk}|$ or $\sum_{i \in I_{jk}} x_i \leq 2\,|I_{jk}|$. This constraint is trivially true and does not restrict x_i. This shows that if Y is a binary vector, x_i will either be constrained to be 0 or only bound by its domain $x_i \in [0,1]$. The maximisation over $\sum_i x_i$ ensures then that x_i automatically takes values in $\{0,1\}$.

For a fixed permutation, we now have a mixed integer linear program (MILP) which is fixed-parameter tractable with respect to the parameter m.

D. Fixed Parameter Tractability

To solve MAX-OPSM we thus have to loop over possible permutations. As for the brute-force algorithm as well as the algorithm of Trapp and Prokopyev, it is not necessary to iterate over all $n!$ permutations. Instead, it is sufficient to iterate only over the permutations π_i induced by the rows i of matrix A. As mentioned above, for each row i we then only have too consider the ordered submatrices of $\hat{A}_i = A([\![n]\!] \setminus [\![i]\!], \pi_i([\![m]\!]))$.

Algorithm 1: Bounded MILP

Require: Matrix $\hat{A}_r \in \mathbb{R}^{\hat{n}_r \times m}$, number of columns γ, and a lower bound l as well as an upper bound u on the number of selected rows.

Ensure : A pair (R, C), such that $\hat{A}_r(R, C))$ is ordered and $|R||C|$ is maximal within the given constraints, or two empty sets

```
X, Y ← solution of mixed integer program (5);
R ← {i + r : X_i > 0} ∪ {r};
C ← {j : Y_j = 1};
return R, C
```

Each loop checks first whether a solution better than the currently best solution may exist and only if this is the case, it starts looking for such a solution. If a better solution is found, it replaces the currently best one. The check is in two steps: The first check skips solving any MILP if previously found solutions imply that there is no solution to the current MILP that can improve over the best solution found so far. The second check solves a bounded MILP whose solution will imply whether the solution to the unbounded MILP will certainly improve over the currently best solution or not.

The bounded and unbounded MILP are instances of the following optimisation problem with parameters γ, u, and l:

$$\operatorname*{argmax}_{X,Y} \sum_i x_i \tag{5}$$

$$\text{s.t.} \begin{cases} y_j + y_k \leq 2 - \dfrac{1}{|I_{jk}|} \displaystyle\sum_{i \in I_{jk}} x_i \text{ for all } j < k \\ \displaystyle\sum_j y_j \geq \gamma \\ X \in [0,1]^{\hat{n}}, Y \in \{0,1\}^m \\ l \leq \displaystyle\sum_i x_i \leq u \end{cases}$$

with $I_{jk} = \{i : \hat{A}(i,j) \geq \hat{A}(i,k)\}$.

Here, γ is the number of columns that should be considered, and u, l are upper, lower bounds on the number of rows, respectively.

The complete subroutine solving the MILP and converting its output is given in Algorithm 1, referred to as `BoundedMILP`$(\hat{A}, \gamma, l, u)$. The relaxed constraints for the row indices together with the upper bound u on their sum, might lead to non-integer solutions. In this case, however, we will

solve another mixed integer program with no upper bound to find the correct solution. The complete algorithm to solve MAX-OPSM is given in Algorithm 2.

Algorithm 2: Quick ILP for MAX-OPSM

Require: Matrix $A \in \mathbb{R}^{n \times m}$
Ensure : An OPSM-pattern (R^*, C^*, π^*), such that $|R||C|$ is maximal

```
1  b ← m;
2  U ← (n − 1, n − 2, . . . , 0) ∈ R^n;
3  for γ ← 3 to m do
4      for i ← 1 to n do
5          if U_i < ⌈b/γ⌉ then go to Line 4 ;
6          C, R ← BoundedMILP(Â_i, γ, 2, ⌈b/γ⌉);
7          if |R| ≥ ⌈b/γ⌉ then
8              C, R ← BoundedMILP(Â_i, γ, ⌈b/γ⌉, U_i);
9          end
10         if |R| > 1 then U_i ← |R|;
11         if |C||R| ≥ b then
12             b ← |C||R|;
13             R* ← R;C* ← C;π* ← π;
14         end
15     end
16 end
17 print π*, R*, C* ;
```

IV. NOISE TOLERANT MAX-OPSM WITH INTEGER PROGRAMMING

We now consider noise-tolerant OPSM-patterns.

Definition 4.1 (Noise tolerant OPSM-pattern (ϵ-OPSM)): For a given noise tolerance $\epsilon \in \mathbb{R}$, a triple (R, C, π) of rows R, columns C, permutation π, induces a *ϵ-noise tolerant order-preserving submatrix* in $A \in \mathbb{R}^{n \times m}$ if for all $i \in R$ and all $j, j' \in C$ it holds that $\pi(j) < \pi(j')$ implies $A(i, j) < A(i, j') + \epsilon$.

Positive tolerance parameters give the level of violation that is tolerated between the selected columns in the OPSM -pattern and negative tolerance parameters give the level stability that is required between the selected columns in the OPSM -pattern. To find MAX-ϵ-OPSM-patterns, for each column pair $j < k$ of a row i, the constraints are now constructed as $I_{jk} = \{i : \hat{A}(i, j) - \hat{A}(i, k) \geq \epsilon\}$.

V. EXPERIMENTS

A. Experimental setup

We ran all experiments on a Intel Core 2 Duo CPU at 3GHz and a total of 4 GB RAM which is considered comparable to or less powerful than the Xeon processor used in [2]. Therefore, we compare our run times against the times reported in their paper. As Trapp and Prokopyev, we also report the run time for varying numbers of columns in the solution

As mentioned above, to cope with the run time demands of their algorithm (TR-K), Trapp and Prokopyev introduced constraints on the search space. Depending on these settings, their algorithm does or does not find the exact optimal solution. In their experiments they ran their algorithm with a sequence of varying constraints, which in many cases would lead to the optimal solution but not always. Hence we report two run times for our algorithm: On the one hand we report the time needed to find a solution of the same quality as reported by Trapp and Prokopyev (henceforth referred to as QuickILP_ρ); on the other hand we report the run time needed to find the exact optimum (henceforth referred to as QuickILP).

We ran a detailed comparison on two real world datasets, as in [2] we consider the Human Gene Expression Index (HuGE) of size 1125×23 and the Breast Cancer Data Set (BRCA) of size 3226×22.

Table I
RUN TIME COMPARISON (IN MINUTES) OF QUICKILP AGAINST THE TR-K IN THE HuGE DATASET. THE NUMBER OF COLUMNS γ IS FIXED TO THE GIVEN VALUES IN ALL ALGORITHMS. ALGORITHM TR-K AND QUICKILP_ρ INCLUDE ALSO CONSTRAINTS ON THE NUMBER OF ROWS SOLVING OPSM.

Cols γ	TR-K (min)	QuickILP_ρ (min)	QuickILP (min)
3	335	<1	45
4	489	<1	91
5	1909	1	180
6	437	4	275
7	524	25	294
8	449	34	421
9	311	40	403

Table II
SIZES OF THE OPSM FOUND BY THE ALGORITHMS IN HuGE DATA. PARAMETER γ CONSTRAINTS THE NUMBER OF COLUMNS IN THE SOLUTION. ALGORITHM TR-K INCLUDES CONSTRAINTS IN ROWS AND THEREFORE IT MIGHT NOT FIND A MAX-OPSM PATTERN. ALGORITHM QUICKILP FINDS ALWAYS A MAX-OPSM PATTERN.

Cols γ	Rows TR-K	Rows QuickILP
3	569	569
4	335	335
5	180	180
6	95	95
7	49	51
8	22	26
9	11	12

The run times for HuGE are reported in Table I and the different solution sizes in Table II. The run times for BRCA are reported in Table III for the constraint search space and in Table IV for the exact solution.

Table III
RUN TIME COMPARISON (IN MINUTES) IN BRCA DATA BETWEEN TR-K AND QUICKILP$_\rho$, WITH CONSTRAINTS ON COLUMNS AND ON ROWS.

Cols γ	Rows ρ	TR-K (min)	QuickILP$_\rho$ (min)
4	347	220	1
	520	586	1
	798	1974	1
6	42	71	4
	63	166	6
	127	2121	36
8	7	17	4
	10	435	10
	14	2370	62

Table IV
RUN TIME TO FIND THE MAX-OPSM FOR THE BRCA DATASET.

Cols γ	quickILP (min)	Rows
3	404	1623
4	653	799
5	1860	354
6	2643	134
7	3408	53
8	2895	21
9	2082	11
10	2359	5
11	713	4

B. *Noise tolerant* OPSM

To study the effect of our noise-tolerance parameter, we used synthetic data. We randomly generate matrices from a normal distribution zero mean and unit variance, selected rows and columns to form a OPSM-pattern (that is, we sorted these entries), and finally perturbed them again by additive noise with zero mean and variance $\sigma \in \{0, 2^{-8}, 2^{-6}, 2^{-4}, 2^{-2}\}$. We performed experiments with different tolerance parameters $\epsilon \in \{0, 2^{-8}, 2^{-6}, 2^{-4}, 2^{-2}\}$. Each experiment is repeated 25 times and we measure the average quality of retrieving the implanted MAX-OPSM by the Jaccard distance:

$$1 - \frac{\left|(\tilde{C} \times \tilde{R}) \cap (C^* \times R^*)\right|}{\left|(\tilde{C} \times \tilde{R}) \cup (C^* \times R^*)\right|}.$$

where C^*, R^* denote the columns and rows selected by the algorithm and $\tilde{C}, \tilde{R}$ denote the columns and rows of the planted OPSM. The results are illustrated in Figure 1.

VI. DISCUSSIONS AND CONCLUSIONS

As motivated by the recent literature, finding exact solutions to biological and medical problems is of crucial importance for the associated applications. In this paper we deal with the problem of finding a largest order preserving submatrix exactly. Building on the solution proposed by Trapp and Prokopyev in [2], we propose here a fixed parameter tractable

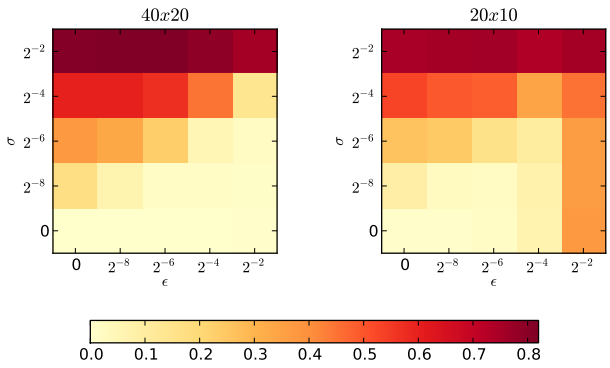

Figure 1. Average Jaccard distance obtained after the 25 rounds for each configuration of our synthetic experiment. Each configuration corresponds to a given value of standard deviation σ (in columns) and ϵ used by the algorithm (in rows). Left plot shows experiments in 40×20 sized matrices; right plot shows experiments in 20×10 sized matrices.

algorithm with respect to the number of columns of the gene expression data. We discuss the complexity analysis and other properties of our algorithm and show how our formulation can be easily extended to mine largest noise tolerant versions of the same problem. An empirical study on two real world datasets confirmed the expected large improvement.

ACKNOWLEDGMENTS

Part of this work was supported by the German Science Foundation (DFG) in the Emmy Noether-Program under 'GA 1615/1-1'. The authors thank Michael Kamp for helpful comments on the paper.

REFERENCES

[1] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. In *Proc. of the 6th annual international conference on Computational biology*, RECOMB '02, pages 49–57, 2002.

[2] A. C. Trapp and O. A. Prokopyev. Solving the order-preserving submatrix problem via integer programming. *INFORMS J. on Computing*, 22:387–400, 2010.

[3] H. Roh and S. Park. A novel evolutionary algorithm for bi-clustering of gene expression data based on the order preserving sub-matrix (opsm) constraint. In *8th IEEE Int Conference on BioInformatics and BioEngineering*, 2008.

[4] B. J. Gao, O. L. Griffith, M. Ester, and S. J. M. Jones. Discovering significant opsm subspace clusters in massive gene expression data. In *2006 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 922–928, 2006.

[5] Q. Fang, W. Ng, and J. Feng. Discovering significant relaxed order-preserving submatrices. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 433–442, 2010.

[6] M. Zhang, W. Wang, and J. Liu. Mining approximate order preserving clusters in the presence of noise, 2008.