Solving the Order-Preserving Submatrix Problem via Integer Programming

Andrew C. Trapp, Oleg A. Prokopyev
Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania 15261, USA,
{act25@pitt.edu, prokopyev@engr.pitt.edu}

In this paper we consider the Order Preserving Submatrix (OPSM) problem. This problem is known to be NP-hard. Although in recent years some heuristic methods have been presented to find OPSMs, they lack the guarantee of optimality. We present exact solution approaches based on linear mixed 0–1 programming formulations, and develop algorithmic enhancements to aid in solvability. Encouraging computational results are reported both for synthetic and real biological data. Additionally, we discuss theoretical computational complexity issues related to finding fixed patterns in matrices.

Key words: biclustering; order-preserving submatrix; data mining; integer programming

---

1 Introduction

Let a data set of m features and n samples be given as a rectangular matrix $A = (a_{ij})_{m\times n}$, where the value a_{ij} is the expression of the i-th feature in the j-th sample. We consider the classification of the samples into classes

$$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r,\ \mathcal{S}_k \subseteq \{1 \ldots n\},\ k = 1 \ldots r,$$

$$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_r = \{1 \ldots n\},\ \ \mathcal{S}_k \cap \mathcal{S}_\ell = \emptyset,\ k, \ell = 1 \ldots r,\ k \neq \ell.$$

This classification should be done so that samples from the same class share certain common properties. Correspondingly, a feature i may be assigned to one of the feature classes

$$\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_r,\ \mathcal{F}_k \subseteq \{1 \ldots m\},\ k = 1 \ldots r,$$

$$\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_r = \{1 \ldots m\},\ \ \mathcal{F}_k \cap \mathcal{F}_\ell = \emptyset,\ k, \ell = 1 \ldots r,\ k \neq \ell,$$

in such a way that features of the class $\mathcal{F}_k$ are "responsible" for creating the class of samples $\mathcal{S}_k$. Such a simultaneous classification of samples and features is called *biclustering* (or *co-clustering*), which can be formally defined as follows [7]:

Definition 1 *A biclustering of a data set is a collection of pairs of sample and feature subsets* $\mathcal{B} = ((\mathcal{F}_1, \mathcal{S}_1), (\mathcal{F}_2, \mathcal{S}_2), \ldots, (\mathcal{F}_r, \mathcal{S}_r))$ *such that the collection* $(\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r)$ *forms a partition of the set of samples, and the collection* $(\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_r)$ *forms a partition of the set of features.*

The criteria used to relate clusters of samples and clusters of features may be of varied nature. In general, biclustering can rely on any common pattern expressed among elements of a bicluster. Biclustering approaches often aim at identifying one bicluster at a time [7]. If more than one bicluster is present in the data set, the biclustering procedure can be repeated in an iterative manner. After each time a new bicluster is found, the corresponding features and samples should be amputated from the data, or their values can be masked with random numbers [8]. Detailed surveys on different types of biclustering patterns, algorithms and related applications can be found in Busygin et al. [7] and Madeira and Oliveira [17].

In this paper we consider the so-called *order-preserving biclusters*. This pattern is introduced in Ben-Dor et al. [4, 5]. Given the data set $A = (a_{ij})_{m \times n}$, the problem is to identify k rows and ℓ columns from the original data matrix in which there exists a permutation of the selected columns such that in every selected row the values corresponding to selected columns are strictly increasing. More formally, let $\mathcal{F}_0$ be a set of row indices $\{f_1, f_2, \ldots, f_k\}$. Then there exists a permutation of a subset of column indices $\mathcal{S}_0 = \{s_1, s_2, \ldots, s_\ell\}$ such that for all $i = 1, \ldots, k$ and $j = 1, \ldots, \ell - 1$ we have that

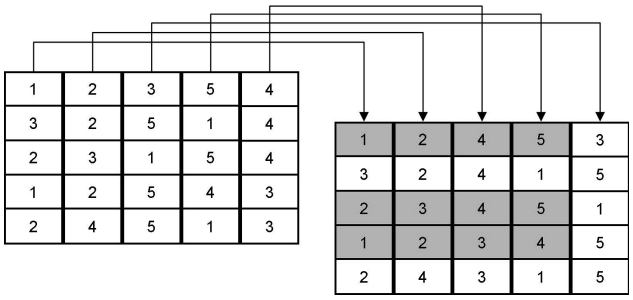


Figure 1: Example of an OPSM.

$$a_{f_i,s_j} < a_{f_i,s_{j+1}}. \tag{1}$$

We call the corresponding submatrix $(\mathcal{F}_0, \mathcal{S}_0) \in \mathbb{N}^{k \times \ell}$ *order-preserving* (OPSM). We provide an exemplary OPSM with respective subsets of column and row indices in Figure 1. The appearance of these types of patterns in real-life data sets has a natural interpretation. Suppose we have a data set for a patient (e.g., DNA microarray), where each sample corresponds to a particular stage of the disease. Then it is quite logical to anticipate that there is a subset of features that are co-expressed with the disease progression. Considering the relative, rather than absolute, orderings of the expression levels gives an indication of the coherent

tendencies, or trends, across the sample set. In fact, we follow the approach of Ben-Dor et al. [5], who consider only the relative ordering (i.e., the ranks) of the expression levels for each gene over permutations of the samples. Using ranks in lieu of exact data values eliminates data scaling issues. A similar situation occurs whenever we consider data representing some temporal progression: data from drug treatment, data from nominally identical exposure to environmental effects, data with some genetic abnormalities, etc. [5, 7].

Recently there has been a growing interest in applying combinatorial optimization based approaches to a variety of biological and medical problems (see, e.g., Balasundaram et al. [2], Busygin et al. [6], Forrester and Greenberg [11], Fung et al. [12], Meneses et al. [18], Tan et al. [24], Trapp et al. [25]). Though some heuristic approaches for solving the OPSM problem are available in the literature [4, 5, 9], in this paper we discuss exact optimization approaches, making use of mixed integer programming to optimally solve the OPSM problem. We first provide a general linear mixed 0–1 programming formulation (Section 3.1) which can be solved using standard solvers such as CPLEX [16]. Secondly, we demonstrate an alternative approach which iteratively solves a series of smaller linear 0–1 programs in conjunction with valid inequalities and other improvements (Section 3.2). Section 4 contains the results of computational experiments on both synthetic (Section 4.1) and real biological (Section 4.2) data with some additional algorithmical enhancements. The following section discusses theoretical computational complexity issues related to the OPSM problem.

2 Computational Complexity Issues

The decision version of the OPSM problem consists of checking whether there exists a $k \times \ell$ order-preserving submatrix $(\mathcal{F}_0, \mathcal{S}_0)$ for given integers k, ℓ and input data matrix $A \in \mathbb{R}^{m \times n}$. More formally, the decision version of the OPSM problem is defined as follows:

Instance: A real-valued data matrix $A = (a_{ij})_{m \times n}$ and two positive integers $k \leq m$ and $\ell \leq n$.

Question: In A, is there an order-preserving submatrix of size k-by-ℓ? That is, we need to check whether there is a set of row indices $\mathcal{F} = \{f_1, \ldots, f_k\}$ and a sequence of columns indices $\mathcal{S} = \{s_1, \ldots, s_\ell\}$ such that for all $1 \leq i \leq k$ and $1 \leq j \leq \ell - 1$

$$a_{f_i, s_j} < a_{f_i, s_{j+1}}. \tag{2}$$

Theorem 1 *[5] The decision version of the OPSM problem is NP-complete.*

In the optimization version of the problem we attempt to find the largest OPSM according to the number of elements $|\mathcal{F}_0|\cdot|\mathcal{S}_0|$, or the weighted sum of the numbers of rows and columns in the resulting submatrix, e.g., $n\cdot|\mathcal{F}_0|+m\cdot|\mathcal{S}_0|$. Due to NP-completeness of the decision version, the optimization version of the OPSM problem is clearly NP-hard. We refer the reader to the classical book by Garey and Johnson [13] for background on computational complexity theory.

We can also look at the computational complexity of the OPSM problem from the point of view of *parameterized complexity theory* [10]. In this theory we say that a problem is *fixed parameter tractable* (FPT) with respect to parameter k if there exists a solution running in $f(k)\times\ell^{O(1)}$ time, where ℓ is the input size of the problem and f is a function of k which is independent of ℓ. In other words, the problem is in FPT with respect to parameter k if it is polynomially solvable for the fixed value of k. Next we show that OPSM is FPT with respect to the number of columns n and FPT with respect to the number of rows m. Though the proofs below are constructive, the enumerative algorithms described there can be utilized for solving OPSM only in the case of *extremely small values* of n or m, respectively.

Proposition 1 *The OPSM problem is polynomially solvable if the number of columns n in input data matrix $A\in\mathbb{R}^{m\times n}$ is fixed.*

Proof. Since n is fixed, we can enumerate all 2^n subsets of the original set of columns indices. For each subset of size r ($r=1,\ldots,n$) we can consider all $r!$ permutations of indices. Then for each row we can check whether the selected permutation of a particular subset of column indices forms an increasing sequence of values. As it was observed in Ben-Dor et al. [5], it can be done in $O(mr)$ time. This results in $\sum_{r=1}^{n}\frac{n!}{(n-r)!}O(mr)$ algorithm for solving the OPSM problem, which is clearly polynomial for each fixed value of n. ■

Proposition 2 *The OPSM problem is polynomially solvable if the number of rows m in input data matrix $A\in\mathbb{R}^{m\times n}$ is fixed.*

Proof. Since m is fixed, we can enumerate all 2^m subsets of the original set of row indices. Then for each subset, assuming that all considered rows are in the resulting submatrix, the objective is to maximize the number of selected columns. Construct a directed graph $G=(\mathcal{N},\mathcal{A})$ as follows. For each column j introduce a node $j\in\mathcal{N}$. Introduce an arc $(j_1,j_2)\in\mathcal{A}$ if and only if $a_{ij_1}<a_{ij_2}$ for every row i in the subset of the considered rows.

The resulting graph G is clearly acyclic. The longest directed path in G then corresponds to the maximum number of columns included in the submatrix. The problem of finding the longest path in an acyclic graph is polynomially solvable (see, e.g., Ahuja et al. [1]), which implies the necessary result. ■

Next, we discuss generalizing the definition of OPSM to finding *any* fixed pattern. For a fixed vector $w = \{w_1, \ldots, w_{n-1}\}$, where $w_j \in \{-1, +1\}$ for all $j = 1, \ldots, n-1$, consider the decision version of the following problem, which is further referred to as the w-OPSM problem:

Instance: A real-valued data matrix $A = (a_{ij})_{m \times n}$ and two positive integers $k \leq m$ and $\ell \leq n$.

Question: In A, is there an order-preserving submatrix of size k-by-ℓ satisfying pattern w? That is, we need to check whether there is a set of row indices $\mathcal{F} = \{f_1, \ldots, f_k\}$ and a sequence of columns indices $\mathcal{S} = \{s_1, \ldots, s_\ell\}$ such that for all $1 \leq i \leq k$ and $1 \leq j \leq \ell - 1$

$$w_j \cdot a_{f_i, s_j} < w_j \cdot a_{f_i, s_{j+1}}. \tag{3}$$

We can observe from (3) that $w_j = 1$ corresponds to up regulation between columns j and $j+1$, (the "up" pattern), whereas $w_j = -1$ corresponds to down regulation between the same columns (the "down" pattern). If $w_j = 1$ for all $j = 1, \ldots, \ell - 1$ then we obtain the above described OPSM problem, looking for a permutation of columns obeying a strictly increasing order. In general, however, the w-OPSM and OPSM problems are not the same. For example, consider the matrix

$$A = \begin{pmatrix} 4 & 5 & 2 \\ 3 & 7 & 6 \end{pmatrix}. \tag{4}$$

If we are looking for the strictly increasing pattern then the largest submatrix consists only of two columns (a single "up" relationship). However, in the case of an {"up","down"} pattern, the final answer is the whole matrix. The question we now ask is whether the w-OPSM problem is difficult for all possible patterns. Is there any pattern w which can be discovered in polynomial time? Unfortunately, it can be shown that for any fixed pattern the problem of finding the largest submatrix satisfying this pattern is NP-hard.

Theorem 2 *w-OPSM is NP-complete for any fixed pattern w.*

Proof. The proof, which we present next is similar in spirit to the result on the OPSM problem and as in Ben-Dor et al. [5] we use the reduction from the *Balanced Complete Bipartite Subgraph* problem, which is known to be NP-complete (see Garey and Johnson [13]):

Instance: Bipartite graph $G = (V, U, E)$, positive integer $K \leq |V|$.
Question: Are there two sets $\bar{V} \subseteq V$, $\bar{U} \subseteq U$ such that $|\bar{V}|=|\bar{U}|=K$ and such that $u \in \bar{U}$, $v \in \bar{V}$ implies that $\{u, v\} \in E$?

For a given pattern w let H be a set of indices such that:

$$H = \{j : \ w_{j-1} = -1, w_j = 1, 1 < j < \ell\} \cup \{1 : \text{ if } w_1 = 1\} \cup \{\ell : \text{ if } w_{\ell-1} = -1\} \qquad (5)$$

Given a bipartite graph $G = (V, U, E)$ let $m = |V|$ and $n = |U| + |H|$. Define the matrix $A = (a_{ij})_{m \times n}$ as follows: *(i)* $a_{ij} = -1$, if $(i, j) \notin E$, *(ii)* $a_{ij} = j$, if $(i, j) \in E$, and *(iii)* $a_{ij} = -1$, if $n - |H| + 1 \leq j \leq n$, where $i = 1, \ldots, |V|$.

Next we show that G contains a balanced complete bipartite subgraph of size $\ell - |H|$ if and only if the matrix A contains an w-order preserving submatrix Q of size $(\ell - |H|)$-by-ℓ.

Assume that there exists an order preserving submatrix of size $(\ell - |H|)$-by-ℓ, which follows pattern w. It can be verified from (3) and (5) that the number of columns, where at least one of the elements is equal to -1, is at most $|H|$. In other words, only columns in positions $j \in H$ in the final submatrix may have elements equal to -1. All other columns can not contain "-1" elements, which implies that we have $\ell - |H|$ columns with only positive elements. By construction, these $\ell - |H|$ rows and $\ell - |H|$ columns with all positive entries will correspond to a complete bipartite subgraph in G.

To show the other direction, assume that there exists a complete bipartite subgraph in G of size $\ell - |H|$. Next we show that the constructed matrix A contains a submatrix Q corresponding to pattern w of size $(\ell - |H|)$-by-ℓ.

Let I and J be the sets of indices which correspond to nodes from $\bar{V}$ and $\bar{U}$ in the complete bipartite subgraph in G, respectively, that is $(i, j) \in E$. We also assume that J is sorted in the increasing order. Let $R = \{|U| + 1, \ldots, |U| + |H|\}$, i.e., R is the set of indices corresponding to columns with all elements equal to -1. Obviously, in Q we keep only rows from A that correspond to nodes from I. Next we construct the respective w-order-preserving submatrix Q according to the following procedure.

a) If $w_1 = 1$, then add column $|U| + 1$ to Q. Remove index $|U| + 1$ from R.

b) If $w_1 = -1$, then let $j^* = \arg\max_{j \in J} j$. Add column j^* to Q and remove j^* from J.

c) For every h such that $1 < h < \ell$:

(1) if $w_{h-1} = 1$ and $w_h = 1$, add $j^* = \arg\min_{j \in J}\{j\}$ to Q and remove j^* from J.

(2) if $w_{h-1} = -1$ and $w_h = -1$, add $j^* = \arg\max_{j \in J}\{j\}$ to Q and remove j^* from J.

(3) if $w_{h-1} = 1$ and $w_h = -1$, add $j^* = \arg\max_{j \in J}\{j\}$ to Q and remove j^* from J.

(4) if $w_{h-1} = -1$ and $w_h = 1$, add any r^* from R to Q and remove r^* from R.

d) If $w_{\ell-1} = 1$, add column $j^* = \arg\min_{j \in J}\{j\}$ to Q and remove j^* from J.

e) If $w_{\ell-1} = -1$, add any column r^* from R to Q and remove r^* from R..

The key intuition is that at every step of the construction we look "ahead" one step and add columns to Q from J or R in a such a way that we will have a column with a smaller or larger value (depending on the given pattern w) at the next step of the construction. If we observe an "up"-"up" pattern (see item "(c)-(1)") in two consecutive columns in given w, then we add to Q a column with the smallest available value from J. Clearly, this implies that we can continue the construction of Q adding columns from J with larger values at the next step of the procedure. Likewise, if we observe "down"-"down" or "up"-"down" patterns (see items "(c)-(2)" and "(c)-(3)"), we add to Q a column with the largest available value from J, allowing the next step of the construction. A similar explanation can be provided for every step in items "(a)", "(b)", "(c)-(4)', "(d)" and "(e)". It is rather easy to verify that the construction is valid and that the obtained submatrix satisfies (3). ■

3 Exact Solution Approaches

Next we discuss exact approaches for solving the OPSM problem which make use of linear 0–1 programming. We first provide a general linear 0–1 programming formulation (Section 3.1). However, our main focus will be on an alternative approach which iteratively solves a series of smaller linear 0–1 programs for a restricted version of the initial OPSM problem, in conjunction with valid inequalities and other improvements (Section 3.2).

3.1 General Integer Programming Formulation

Our first approach is a general linear 0–1 programming formulation to the OPSM problem. Using the simple concept of filling ordered positions with columns from input matrix A, we attempt to find a permutation of the original columns so that the order induced across a subset of rows is strictly increasing; of all such column and row subsets, we wish to find the largest such submatrix. That is, we assume there are $1, \ldots, n$ positions that may be filled by any of the n columns. We then attempt to fill the first $K \leq n$ positions with columns such that all included rows maintain this increasing order.

To facilitate this concept, we introduce a binary variable s_{jk} for each column j and possible position k, with $s_{jk} = 1$ implying that column j has been selected for position k. Figure 2 illustrates the relationship between s_{jk} variables and column-position interactions (for simplicity, in this illustration we assume that all rows are included).

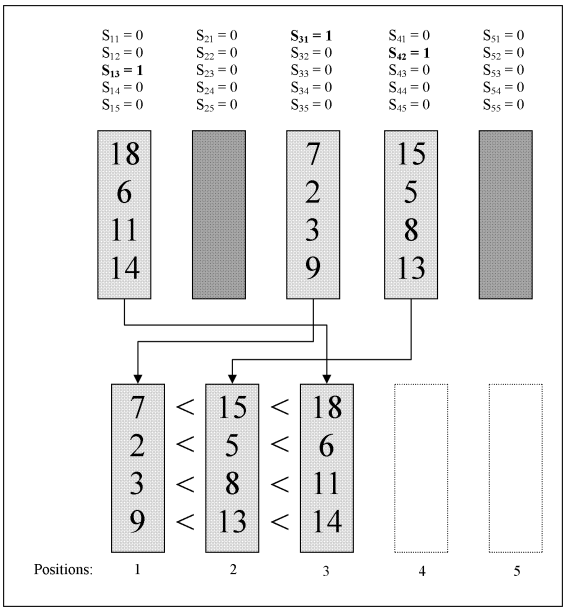


Figure 2: Relationship between s_{jk} variables and column-position interactions

Additionally, for each row we introduce a binary variable x_i, with $x_i = 1$ indicating that row i is selected for the OPSM. Likewise, for each column we introduce a binary variable y_j, with $y_j = 1$ implying that column j is selected in the OPSM. Finally, we define binary variables z_{ij} as the product of each row and column combination, that is, $z_{ij} = x_i y_j$. Thus, one binary z_{ij} variable corresponds to each entry in the input matrix A. We use these variables to construct GF-OPSM below, which is a valid formulation for the OPSM problem.

$$\text{(GF-OPSM)}: \ \max \sum_i \sum_j z_{ij} \tag{6}$$

subject to

$$x_i + \sum_{u=1}^{k} s_{ju} + \sum_{v=k+1}^{n} s_{\ell v} \leq 2 \quad \forall\, j, \ell \text{ such that } a_{ij} \geq a_{i\ell},\ i = 1, \ldots, m,\ k = 1, \ldots, n-1 \tag{7}$$

$$\sum_{j=1}^{n} s_{jk} \geq \sum_{j=1}^{n} s_{j,k+1}, \quad k = 1, \ldots, n-1 \tag{8}$$

$$\sum_{k=1}^{n} s_{jk} = y_j, \quad j = 1, \ldots, n \tag{9}$$

$$\sum_{j=1}^{n} s_{jk} \leq 1, \quad k = 1, \ldots, n \tag{10}$$

$$z_{ij} \leq x_i, \; z_{ij} \leq y_j, \quad \forall \; i, j \tag{11}$$

$$x_i \in \{0,1\}, \; y_j \in \{0,1\}, \; z_{ij} \in \mathbb{R} \;\; \forall \; i, j \tag{12}$$

The objective function of this formulation ensures that we find the *largest* OPSM by maximizing the number of entries included in the OPSM. Constraint set (7) enforces the condition of strictly increasing order by requiring that, if row i is in the final solution, and $a_{ij} \geq a_{i\ell}$ holds, then column j cannot appear to the left of column ℓ in the final permutation of columns. Constraints (8) ensure that, if position k is not filled in the final permutation (i.e., there are less than k columns in the final submatrix), then positions $k+1, \ldots, n$ are also not filled in the final permutation. These constraints enforce a decision hierarchy which removes symmetry (and thus duplicate solutions) from the problem. Constraints (9) ensure that, if column j is chosen, it fills exactly one position in the final permutation. On the other hand, constraints (10) ensure that at most one column can fill any one position in the final permutation. Finally, constraints (11) ensure that element (i, j) may be in the final OPSM if and only if both row i and column j are included.

Formulation GF-OPSM has $m + n + mn + n^2$ variables in total, of which $m + n + n^2$ are binary, and at most $2n + 2mn + (n-1) + \frac{1}{2}mn(n-1)^2$ constraints. Unfortunately, this formulation did not perform well in computational testing. Since our main focus will be on the algorithm which we describe next, we leave tightening of the GF-OPSM formulation to future research.

3.2 Iterative Algorithm

The key idea of this approach is to iteratively solve a series of smaller linear 0–1 programs for a restricted version of the initial problem. We also derive valid inequalities and demonstrate other enhancements for the proposed algorithm.

3.2.1 Basic IP Formulation

Suppose we are searching for the largest submatrix exhibiting a simple increasing pattern, where column permutation is not allowed. Given this scenario, the OPSM problem simplifies to simultaneously finding:

- a set of row indices $\{f_1, f_2, \ldots, f_k\} \subseteq \mathcal{F}_0$,
- a set of column indices $\{s_1, s_2, \ldots, s_\ell\} \subseteq \mathcal{S}_0$, such that $s_1 < s_2 < \ldots < s_\ell$, and
- for all $i = 1, \ldots, k$ and $j = 1, \ldots, \ell - 1$ we have that $a_{f_i,s_j} < a_{f_i,s_{j+1}}$; that is, we do not permit *inversions* on the values corresponding to the set of column indices on included rows.

These conditions motivate our construction of the next formulation, which we will refer to Compact-Formulation, or CF-OPSM. As in GF-OPSM, we introduce binary variables x_i for each row, binary variables y_j for each column, and binary variables z_{ij} for each row and column combination. Given this setup, finding the largest submatrix obeying the strictly increasing pattern is tantamount to solving the following 0–1 programming problem:

$$\text{(CF-OPSM)}: \ \max \sum_i \sum_j z_{ij} \tag{13}$$

$$\text{s.t.} \ \ z_{ij} + z_{ik} \leq x_i \ \ \forall\, j < k \text{ and } a_{ij} \geq a_{ik}, \ i = 1, \ldots, m \tag{14}$$

$$z_{ij} \geq x_i + y_j - 1, \ z_{ij} \leq x_i, \ z_{ij} \leq y_j, \quad \forall\, i, j \tag{15}$$

$$x_i \in \{0, 1\}, \ y_j \in \{0, 1\}, \ z_{ij} \in \{0, 1\} \ \ \forall\, i, j \tag{16}$$

The objective function of CF-OPSM is identical to that used in GF-OPSM, which aims to find the largest OPSM by area. The first constraint set (14) ensures that, for any of included row i's entries taken pairwise, with both $j < k$ and $a_{ij} \geq a_{ik}$, then at most one of the two z_{ij} entries may be included in the OPSM. That is, for any row i included in the OPSM, constraints (14) prevent *inversions* on included columns. An alternative view of these constraints is that, when row x_i is included in the OPSM, they represent pairwise clique inequalities, and represent both necessary and sufficient conditions to characterize the CF-OPSM polyhedron. Constraint set (15) compels the z_{ij} variables to take value 1 whenever both row i and column j are selected to be in the OPSM. Formulation (13)-(16) has $m + n + mn$ binary variables and at most $3mn + \frac{1}{2}mn(n-1)$ constraints.

3.2.2 Basic Iterative Algorithm

The main intuition behind our algorithm is as follows. Suppose the first row of A is in the final solution. If we then permute the columns of A so that the elements of the first row appear in increasing order, then the solution of CF-OPSM, together with the constraint $x_1 = 1$ will provide the largest OPSM which includes row 1. Repeating this approach on each subsequent row gives the largest submatrix for each respective row. Afterwards, assuming there is a single largest OPSM (we address the possibility of multiple optimal solutions later), then k rows will share the largest objective value; these rows constitute $\mathcal{F}_0$, and together with the set of included columns $\mathcal{S}_0$ form the largest OPSM of A. This motivates the following iterative algorithm.

Algorithm 1 *[Basic Iterative Algorithm]*

1. *Assign* $h := 1$.
2. *Permutate columns of* A *such that the entries in row* h *occur in increasing order; call new matrix* $\hat{A}_h$.
3. *Generate formulation CF-OPSM for matrix* $\hat{A}_h$. *Add additional constraint* $x_h = 1$.
4. *Solve the obtained linear 0–1 formulation. Let* Z^h *be the obtained optimal objective function value, and store off the optimal solution for row* h.
5. *If* $h < m$, *let* $h := h + 1$ *and go to 2.*
6. $Z^* = \max_h Z^h$.
7. *Return* Z^*, *its corresponding optimal solution, and STOP.*

For each row h, Algorithm 1 formulates and solves an instance of the CF-OPSM integer program, identifying the largest submatrix having an strictly increasing pattern according to permuted matrix A_h. Since we iteratively consider each row $h = 1, \ldots, m$, Algorithm 1 finitely terminates. At its conclusion, after searching over all rows $h = 1, \ldots, m$, the largest OPSM corresponding to input matrix A is provided as output.

3.2.3 Valid Inequalities

The pairwise constraints (14) are reminiscent of classical pairwise clique inequalities. Indeed, while it is known that pairwise clique inequalities are sufficient to represent the independent set polyhedron, they typically are not facet-defining [21]. However, these pairwise inequalities

can be strengthened into facet-defining inequalities by identifying the *maximal* clique to which a given node belongs.

In a similar manner, pairwise inversion inequalities (14) can be strengthened. In general, if for row i the relationship $a_{ij_1} \geq a_{ij_2} \geq \ldots \geq a_{ij_k} \geq \ldots \geq a_{ij_\ell}$ holds, where $j_1 < j_2 < \ldots < j_k < \ldots < j_\ell$, then the corresponding inversion inequality

$$z_{ij_1} + z_{ij_2} + \ldots + z_{ij_k} + \ldots + z_{ij_\ell} \leq x_i \tag{17}$$

is valid to impose on CF-OPSM. The set $a_{ij_1}, a_{ij_2}, \ldots, a_{ij_k}, \ldots, a_{ij_\ell}$ defines a *maximal* decreasing subsequence among the elements of row i if we cannot augment the current decreasing subsequence with an additional element. In this case we will refer inequality (17) as a *maximal inversion* inequality.

Theorem 3 *Maximal inversion inequalities are facets of the convex hull of integer solutions to the CF-OPSM polyhedron.*

Proof. For a given row i, let $C = \{j_1, j_2, \ldots, j_k, \ldots, j_\ell\}$ correspond to the column indices representing a maximal inversion of size ℓ on row i. Now if row i is selected for any optimal OPSM, then at most one of the elements j_k may be included in the OPSM. Thus the maximal inversion inequalities (17) are valid inequalities for the CF-OPSM polyhedron. To demonstrate that maximal inversion inequalities are facet-defining for the CF-OPSM polyhedron, which has full-dimension, we need to identify $mn + m + n$ affinely independent, feasible points that satisfy the maximal inversion inequality at equality. Such a set of vectors can be constructed as follows.

To start, the origin trivially satisfies (17) at equality, and is feasible, and so is one such point. Also, the vectors consisting of $y_j = 1,\ j = 1, \ldots, m$, and all other components zero, give m more such vectors. Now for row i containing our maximal inversion, we can construct $\ell \leq m$ additional vectors using each of the ℓ elements of the maximal inversion by setting $x_i = 1$, $y_{j_k} = 1$, and $z_{ij_k} = 1$, with all other components zero. Let column j_q correspond to one of the $m - \ell$ elements not in the columns of the maximal inversion C. For each such column j_q, we can construct an additional vector by taking $x_i = 1$, $y_{j_1} = 1$, $z_{ij_1} = 1$, $y_{j_q} = 1$, and $z_{ij_q} = 1$. In this manner we can construct $m - \ell$ additional affinely independent, feasible vectors satisfying (17) at equality. Finally, for each row $h \neq i$ ($n - 1$ total), we can construct $(m + 1)$ additional vectors as follows. One such vector is $x_h = 1$, with all other components

zero; the other m vectors have the form $x_h = 1$, $y_j = 1$, and $z_{hj} = 1$ for $j = 1, \ldots, m$. Thus this last step constructs an additional $(n-1) \times (m+1) = mn - m + n - 1$ such vectors. In conclusion, after identifying $1 + m + \ell + (m - \ell) + nm - m + n - 1 = mn + m + n$ total affinely independent, feasible vectors satisfying (17) at equality, we conclude that maximal inversion inequalities (17) are indeed facets of the CF-OPSM polyhedron. ■

Since the maximal inversion inequalities (17) define facets of the CF-OPSM polyhedron, this leads to the natural question of how to quickly find such relationships, i.e., maximal decreasing subsequences. Schensted [22] gave an $O(n\ log\ n)$ algorithm, utilizing binary search, to solve the longest decreasing subsequence (LDS) problem. Further in the paper, we will refer to this algorithm as *FindLDS*.

The longest decreasing subsequence for a particular row in the original data matrix constitutes a maximal decreasing subsequence, which we utilize to construct the respective facet-defining valid inequality. While finding the longest decreasing subsequence is beneficial, other maximal decreasing subsequences of the same or smaller size may exist, which *FindLDS* does not identify. We therefore extend this algorithm to efficiently locate the maximal decreasing subsequence passing through *each* element of the given sequence.

For all elements $j = 1, \ldots, n$, apply Algorithm *FindLDS* and find the longest decreasing subsequence up to and including element j; store off this subsequence. Now apply the same algorithm in reverse (*FindLIS*), starting with element n, stepping through the FOR loop in reverse, until element j is reached. This will generate the longest *increasing* subsequence up to and including element j. Again, store off this subsequence. Combining the longest decreasing subsequence up to and including element j on a forward pass, together with the longest increasing subsequence up to and including element j on a reverse pass, gives the longest decreasing subsequence through each element $j = 1, \ldots, n$.

For each element j, Algorithms *FindLDS* and *FindLIS* take $O(n\ log\ n)$ time to complete. With n elements in the sequence, this implies $O(n^2\ log\ n)$ time. Since in the worst case we consider sequences corresponding to rows $h = 1, \ldots, m$, the above described approach to find maximal decreasing subsequences has an overall run time of $O(mn^2\ log\ n)$.

3.2.4 Nodal Constraints

Murray and Church [19, 20] make another interesting observation regarding maintaining the necessary restrictions that pairwise clique inequalities impose. In particular, on a given graph

G they propose an alternative set of valid inequalities called *nodal* constraints which can effectively replace pairwise clique inequalities. They define nodal constraints in the context of a graph G, where a constraint for each node n_i is generated based on its neighborhood of adjacent nodes. We can easily extend this definition to our context of sequences, where for a given row i of A and its elements $a_{i1}, \ldots, a_{ij}, \ldots, a_{in}$, we define the *neighborhood* N_{ij} of element a_{ij} as the set of all elements a_{ik} such that either: (i) $k > j$ and $a_{ik} \leq a_{ij}$, or (ii) $k < j$ and $a_{ik} \geq a_{ij}$. Based on this definition, for row i we introduce nodal constraints for each element a_{ij} as

$$n_{ij} z_{ij} + \sum_{k \in N_{ij}} z_{ik} \leq n_{ij} x_i \; \forall \; j, \tag{18}$$

where we set the value n_{ij} as $n_{ij} = |N_{ij}|$. The nodal constraint (18) for element a_{ij} efficiently represents all of the inversion restrictions that the pairwise clique inequalities imply. This is because, if variable $z_{ij} = 1$ for element a_{ij}, then constraint (18) forces the non-negative variables z_{ik} corresponding to all neighbors $a_{ik} \in N_{ij}$ to be zero. However, if $z_{ij} = 0$, then no restrictions are forced upon the neighbors of element a_{ij}.

For a given row i of A there are exactly n nodal constraints, one for each column. Thus there are exactly mn total nodal constraints per CF-OPSM instance. Considering that the number of inversions on a typical row of A is $O(n^2)$, using inequalities (18) in place of (14) greatly reduces the total number of constraints and corresponding size of the model, especially since a given CF-OPSM formulation has inversion restrictions for $O(m)$ rows. Our computational testing reveals that the CF-OPSM formulation using nodal constraints maintains almost all of the tightness of the original CF-OPSM formulation. We discuss this further in Section 4.1.3.

3.2.5 Further Enhancements

Next we present additional strategies aimed at improving the solution time of Algorithm 1.

Turning Off Previous Rows. For current row h, one such observation is that, for all previous rows $i = 1, \ldots, h-1$, we have $x_i = 0$. This is a valid inequality because previously solved row i is either: (i) not in the OPSM; or (ii) in the OPSM, but since we have already identified the optimal solution (OPSM) for row i, it is not necessary to locate that solution again. Thus, in either case, $x_i = 0$ is a valid inequality for all rows $i = 1, \ldots, h-1$.

Valid Lower Bounds. Another observation is that, for any row $h > 1$, a valid lower bound on the objective value for all future rows is $\bar{Z} = \left(\max_{k=1,\ldots,h-1} Z^k \right) + 1$. That is, in order to

locate a larger OPSM than the best found in previous rows, the current objective value Z^h must outperform $\bar{Z}$. Thus the inequality

$$\sum_i \sum_j z_{ij} \geq \bar{Z} \tag{19}$$

is valid for all rows $h > 1$. If for current row h no feasible solution exists having objective value $Z^h \geq \bar{Z}$, we are free to move on to the next row. To allow for *multiple* optimal solutions in our code, simply removing the increase of 1 in the definition of $\bar{Z}$ will permit this possibility. Furthermore, CPLEX 11 [16] offers the solution pool feature, which has the capability to find all solutions within a certain user-defined tolerance of optimality, as well as to find diverse solutions to MIPs. Thus, this feature could be used to find and store all optimal OPSMs.

LP-based Pruning. It is well-known that linear programming relaxations generally solve much more quickly than their integer counterparts. With this in mind, at each iteration after generating the necessary IP formulation, we solve its LP relaxation by allowing $x_i \in [0, 1]$, $y_j \in [0, 1]$, and $z_{ij} \in [0, 1]$. Solving this LP relaxation, denote the optimal objective value as $Z^*_{h_{LP}}$. If the obtained LP bound $Z^*_{h_{LP}}$ does not at least match $\bar{Z}$, do not proceed solving the corresponding 0–1 programming problem. As the integral optimal solution to this instance cannot yield a larger optimal objective value than $Z^*_{h_{LP}}$, i.e., $Z^*_h \leq Z^*_{h_{LP}} \leq \bar{Z}$, we are free to move on to the next row.

LP-based Presorting. For *every* row $h = 1, \ldots, m$, generate and solve the m LP relaxations of the respective IP formulations and record the corresponding optimal objective values. Re-sort the rows of the original data matrix A in the following manner. In the first row place the row corresponding to the largest optimal objective value. Order all remaining rows $2, \ldots, m$ into increasing order corresponding to their recorded optimal objective values. Knowing that the row with the highest LP relaxation optimal objective value gives an upper bound on the largest OPSM, placing this row first will presumably give the algorithm the best chance to find a row contained in the largest actual OPSM, thereby achieving the largest possible initial solution $\bar{Z}$.

Re-sorting the remaining rows into increasing order does two additional things. First, when combined with the valid inequality (19), this often forces the instances of IP formulations for rows subsequent to row 1 to become infeasible, as they likely cannot find a feasible solution satisfying such a restrictive inequality. Thus, little time is typically spent on such

rows. Secondly, when this step is combined with the pruning process above, often the algorithm also prunes quickly, thereby providing additional savings on computation time. This approach proves very useful in our computational experiments.

Stopping Criteria. As we described above we set $x_i = 0$ for all rows $i < h$, and so a valid upper bound on the largest potential objective function value for row h is given by $W = n \times (m - h + 1)$, corresponding to $z_{ij} = 1 \; \forall \; j, \; i = h, \ldots, m$. Thus, if the objective value of the current best solution $\bar{Z} \geq W$, we can safely terminate the algorithm; it is not possible for a larger submatrix to exist in the remaining rows.

3.2.6 Enhanced Iterative Algorithm

We next incorporate many of these improvements into the following enhanced iterative algorithm for solving the OPSM problem.

Algorithm 2 *[Enhanced Iterative Algorithm]*

1. *Perform LP-based presorting of rows of A utilizing either pairwise inequalities (14) or nodal constraints (18) in the respective LP formulations. Assign* $h := 1$.

2. *Permutate columns of A such that the entries in row h occur in increasing order; call new matrix* $\hat{A}_h$.

3. *Generate formulation CF-OPSM for matrix* $\hat{A}_h$ *using either pairwise inequalities (14) or nodal constraints (18).*

3(a). *Add additional constraint* $x_h = 1$ *and* $x_k = 0$ *for all* $k < h$.

3(b). *Generate maximal decreasing subsequences via modified FindLDS and FindLIS Algorithms (see discussion in Section 3.2.3); add respective valid inequalities (17).*

3(c). *If* $h > 1$ *add a valid lower bound (19).*

3(d). *If* $h > 1$ *solve LP relaxation of the respective IP formulation using either pairwise inequalities (14) or nodal constraints (18). Let* $Z^*_{h_{LP}}$ *be the obtained optimal objective function value. Go to 6 if* $Z^*_{h_{LP}} \leq \bar{Z}$.

4. *Solve the obtained IP formulation. Let* Z^h *be the obtained optimal objective function value, and store off the optimal solution for row h.*

5. *Let* $\bar{Z} = \max_{k=1,\ldots,h} Z^k$.

6. *If* $h < m$, *let* $h := h + 1$. *Check* ***Stopping Criteria*** *and go to 2, if necessary.*

7. *Return $Z^* = \bar{Z}$, its corresponding optimal solution, and STOP.*

4 Computational Experiments

We designed test experiments using both synthetic and real biological data to verify the effectiveness of the proposed methods, with our findings from synthetic data testing aiding in our choice of the best configuration for testing real data. To perform the optimization, we used the callable library of CPLEX 11.0 [16]. All algorithms are coded in C++. This section discusses our findings.

4.1 Experiments with Synthetic Data

4.1.1 Setup and Data

Our testing environment consisted of a Windows XP machine equipped with a 2.4GHz Pentium 4 processor plus 2GB of RAM.

Algorithmic Parameters. STOP [3], an automated tool to tune optimization software parameters, provided us with a suite of parameters that gave us marked performance improvements. Specifically, we adjusted the CPLEX default parameter settings for the MIP Emphasis parameter to *Feasibility over Optimality*, the Implied Bound, Clique, and Gomory Fractional Cuts parameters to *Aggressive*, the Variableselect (Branching) parameter to *Strong Branching*, and the RINS Heuristic parameter to *Every Iteration*.

Moreover, we *always* implemented the **Stopping Criteria** from Section 3.2.5. Additionally, in Steps **1** and **3(d)** of Algorithm 2, we use the CPLEX barrier algorithm, an interior-point method, rather than the simplex algorithm (which proved to be less efficient for our larger instances) to perform the LP-relaxation optimization. Since only the optimal objective values of these relaxations are necessary for our methods, and not the optimal solutions themselves, we also turn off the expensive cross-over routines (BARCROSSALG), thereby recovering additional computational savings.

Synthetic Data Generation. In order to create our synthetic data, we coded a test instance generator according to the procedure outlined in Ben-Dor et al. (2003). This generator plants an order-preserving submatrix into an otherwise randomly generated data matrix. It first randomly chooses the indices for planted rows and columns. Subsequently, it randomly chooses an ordering for planted columns. Finally, the generator randomly assigns ranks to the data matrix in a way which is consistent with the planted submatrix. The

Table 1: Variations of tested algorithms: Algorithm **A** is the General Formulation (6) – (12), while Algorithms **B** through **M** are variations of the Iterative Algorithm.

	Algorithmic Variations												
Formulation / Enhancement	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
General Formulation	•												
All Pairwise Ineqs. (14)		•	•	•	•	•	•					•	•
Nodal Ineqs. (18)								•	•	•	•		
Turning Off Previous Rows				•	•	•	•	•	•	•	•	•	•
Max Inversion Ineqs. (17)						•	•	•	•	•	•	•	•
Valid LB Ineq. (19)										•	•	•	•
LP-based Presorting			•		•		•		•		•		•

input is the number of rows m and columns n of A, the number of columns s in the planted OPSM, and the probability p of a given row being included in the submatrix to be planted. Using this information the test instance generator randomly embeds a submatrix according to the input parameters specified, while simultaneously identifying (for the user's benefit) the *actual* submatrix planted within A.

4.1.2 Test Scenarios for Synthetic Data

We propose two test scenarios in order to determine the best algorithm configuration for further testing of real data.

Test Scenario: Vary Algorithms. The first test scenario evaluates the performance of variations of our algorithm on some smaller test instances. Specifically, there are 6 levels of test sets, with data matrix sizes (m, n) ranging from $(20, 10)$ to $(50, 20)$; for each level, there are three test instances. The embedded OPSMs comprise approximately 25% of the overall data matrix size; that is, we set $p = .5$ and $s = .5n$, to give a mean embedded OPSM size of $p \times m \times s = .25mn$. Regarding algorithms, we first test the General Integer Programming Formulation (Section 3.1) and record the run times for CPLEX 11.0 to locate an OPSM of optimal size on the generated test instances. We compare these results against variations of the iterative algorithm (Section 3.2); starting with Algorithm 1 (Basic Iterative Algorithm), we sequentially augment this algorithm with valid inequalities and further enhancements, until reaching Algorithm 2 (Advanced Iterative Algorithm). Run times are recorded for the amount of time necessary to find an optimal OPSM for each algorithm and test instance combination, and as there are three test cases, where possible we provide the mean run time

for each test level. The algorithmic variations are reported in Table 1.

Test Scenario: Vary OPSM Size. The second test scenario is to vary the size of the planted OPSM. For the two mean embedded OPSM sizes of $.25mn$ and $.2mn$ (i.e., 25% and 20% of the overall size of data matrix A), we create 15 levels of test sets, with each test set again having three test instances. Algorithm 2 (Advanced Iterative Algorithm, version **K**, which proves to be the best variation) is run on these test levels and instances to determine how changes in input data size affect our algorithm.

Table 2: Comparison of algorithmic run times on test instances, reporting m, n, and optimal OPSM size Z^*. All times are in seconds, with the fastest run times in bold. Where possible, every fourth line details the average of the previous three lines. An entry of "–" indicates that the run time of the specific instance, or average, was not within a factor of 5 of the fastest algorithm.

m	**n**	Z^*	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
20	10	50	–	–	–	–	–	24.5	10	17.8	**7.9**	17.3	8.3	24.3	9.9
20	10	50	–	–	–	–	–	14.7	13.8	14.8	12.1	4.9	12	**4**	13.7
20	10	40	–	–	–	161.3	175.4	94.6	125.2	150.5	139.8	87.6	**42.5**	48.6	50.8
Average			–	–	–	87.4	–	44.6	49.7	61.1	53.3	36.6	**21**	25.6	24.8
30	10	70	–	–	–	–	–	–	–	–	–	21.6	23.5	**9.5**	20.2
30	10	85	–	–	–	–	–	–	8.9	–	**8.6**	–	8.9	–	8.6
30	10	50	–	–	–	–	–	–	–	–	–	169.1	207.1	**78.5**	136.8
Average			–	–	–	–	–	–	–	–	–	98.5	79.8	60.8	**55.2**
30	15	144	–	–	–	–	–	2.8	–	2.8	–	1.3	–	**1.2**	–
30	15	64	–	–	–	–	–	–	–	–	–	2106.8	**466.7**	1074	828.6
30	15	144	–	–	–	–	–	3	–	2.6	–	1.5	–	**1.4**	–
Average			–	–	–	–	–	–	–	–	–	703.2	**160.8**	358.9	281.7
40	15	160	–	–	–	–	–	8.5	11.8	6.9	11.9	3.7	11.5	**3.4**	12.3
40	15	168	–	–	–	–	–	9.3	12.4	8.8	13.1	5.1	13.4	**4.7**	13
40	15	168	–	–	–	–	–	–	**12.3**	–	12.4	–	12.8	–	12.8
Average			–	–	–	–	–	–	**12.2**	–	12.5	–	12.6	–	12.7
40	20	210	–	–	–	–	–	–	16.7	–	**14.7**	–	15.6	–	17.3
40	20	230	–	–	–	–	–	12.5	15.8	9.4	14.7	3.6	15.5	**3.3**	–
40	20	220	–	–	–	–	–	9.9	–	10.3	14.7	4	–	**3**	–
Average			–	–	–	–	–	–	16.2	–	**14.7**	–	15.6	–	17.2
50	20	210	–	–	–	–	–	162.5	146.6	181.8	155.9	99	156.8	**79.9**	147.6
50	20	200	–	–	–	–	–	–	140.2	–	234.9	–	233.7	–	**117**
50	20	170	–	–	–	–	–	–	–	–	–	–	**967.9**	–	1648.9
Average			–	–	–	–	–	–	–	–	–	–	**452.8**	–	637.8

4.1.3 Synthetic Data: Results and Discussion

We present the results of the first test scenario in Table 2. The fastest run times for a given test instance (as well as the averages) are bolded, and any run time which was not within 5 times of the fastest run time is indicated by "–". It is immediately clear that the General Formulation, along with Algorithms **B**, **C**, **D**, and **E**, cannot compete with the more advanced iterative algorithms. Algorithms **F** and beyond are more competitive, coinciding with their inclusion of the facet-defining maximal inversion inequalities (17).

One general trend is that, holding all else the same, test instances of equal m and n having larger embedded OPSMs tend to solve more quickly than those with smaller OPSMs. This is evident, for example, in the second test instance of size $m = 30$ and $n = 15$; the optimal OPSM size of Z^* was 64, much less than the expected size of $.25mn = 112.5$. Correspondingly, the run times for this instance are much larger than the first and third test instances for the same levels of (m, n).

Furthermore, the results of Table 2 indicate that the fastest algorithm is one of Algorithms **K**, **L**, and **M**. Algorithm **L** has reasonable run times only for those test instances which happen to have their first row in a rather large OPSM. For such instances, since Algorithm **L** does not perform **LP-based Presorting** and does include the **Valid LB Inequality** (19), it has a favorable initial optimal objective value from the first row, uses the lower bounding inequality (19), but does not include the additional overhead of performing the **LP-based Presorting** step. In general, however, since there is no way to know whether the first row is included in a rather large OPSM, we prefer to use the **LP-based Presorting** step present in Algorithms **K** or **M**. This is because the additional computational time of solving m LP relaxations (via the efficient barrier algorithm) and sorting the rows accordingly almost always recovers that time by finding a first row with a large (sometimes optimal) OPSM.

Algorithms **K** and **M** seem roughly comparable, though we slightly prefer Algorithm **K**, since it has 6 of the fastest run times, while **M** reflects only two. More notable, however, is on difficult test instances where the value of Z^* is much less than the expected size of $.25mn$ (e.g., instance 2 of $(30, 15)$ and instance 3 of $(50, 20)$), Algorithm **K** has the fastest run times. Indeed, this trend continues as the values of m and n grow, with the benefits of the smaller nodal inequality formulation in Algorithm **K** outweighing the slightly tighter formulation (but increasingly unwieldy due to greater constraint matrix expansion) given by the pairwise inversion inequalities in Algorithm **M**.

Furthermore, the nodal inequality formulation of Algorithm **K** does not adversely impact Step **1** (Presorting), since *each* of the m subproblems incurs a small gain in the LP relaxation optimal objective value. Thus the relative rankings of the optimal objective function values, needed for Step **1**, remained more or less constant over all rows.

Regardless of whether an algorithm uses **LP-based Presorting**, the most time consuming step is to repeatedly solve integer programs. This is why Step **1** (Presorting) is in general advantageous, because after solving the first integer program, we use its optimal solution as a bound for future integer programs, serving to avoid the solution of additional integer programs (as detailed in Section 3.2.5, **Valid Lower Bounds** and **LP-based Pruning**).

Table 3: Two types of embedded OPSMs (25% and 20% of matrix A). Run times (in minutes) represent averages over 3 test instances.

Data		**Algorithm K**	
Rows	**Cols**	**25%**	**20%**
20	10	< 1	< 1
30	10	1	1
30	15	2	1
40	15	< 1	2
40	20	< 1	12
50	20	4	24
100	20	2	8
100	30	2	19
100	40	4	3
100	50	5	6
200	50	29	32
400	50	139	166
600	50	286	337
800	50	503	546
1000	50	850	916

Table 3 displays the results of the second test scenario. It highlights the trend that Algorithm **K** has an easier time locating OPSMs of *larger* expected size. Also, we should mention that the General Formulation does not perform well in identifying OPSMs in even modest size input matrices; the maximum size it was able to handle in a reasonable amount of time was $(30, 10)$, in around 2 hours of computation.

4.2 Experiments with Real Data

4.2.1 Setup and Data

Our testing environment consisted of a Windows XP machine equipped with Dual-Core Intel Xeon processor with 3GHz processor and 3GB of RAM. In our experiments we considered two real data sets, which we briefly describe below.

Breast Cancer Data Set (BRCA). The BRCA breast tumor data set has $m = 3226$ genes (rows) and $n = 22$ samples (columns). Of these samples, 8 have brca1 mutations, another 8 have brca2 mutations, and the remaining 6 represent sporadic breast tumors. We refer the reader to Hedenfalk et al. [14] for further details regarding this data set. This data set was utilized by Ben-Dor et al. [5].

Human Gene Expression Index (HuGE). Our second data set was the Human Gene Expression (HuGE) Index data set [2009]. The purpose of the HuGE project is to provide a comprehensive database of gene expressions in normal tissues of different parts of human body and to highlight similarities and differences among the organ systems. The data set consists of 59 samples from 19 distinct tissue types, and was obtained using oligonucleotide microarrays capturing 7070 genes. We removed all rows containing at least one incomplete or missing entry. Additionally, we retained samples from three of the main tissues (Brain, Liver, and Muscle), leaving 1125 genes and 23 columns. This particular data subset was utilized for experiments on biclustering in Trapp et al. [25]; other experiments with HuGE appear in Busygin et al. [6].

4.2.2 Detecting Statistical Significance

An important contribution in Ben-Dor et al. [4] is the derivation of *statistically significant* OPSM patterns. As they explain, for a set $T \subset \{1, \ldots, n\}$ of columns of size γ together with a linear ordering $\pi = (t_1, t_2, \ldots, t_\gamma)$, the probability that a random row supports a given model (T, π) is $(1/\gamma!)$. Further, since the rows are assumed as independent, the probability of having at least ρ rows supporting model (T, π) is the ρ-tail of the $(m, (1/\gamma!))$ binomial distribution. Since there are $n \cdot (n-1) \cdots (n-\gamma+1)$ ways to choose a complete model of size γ, they provide an upper bound on the probability of having a model of size γ with at least ρ rows as

$$U(\gamma, \rho) = n \cdots (n-\gamma+1) \sum_{i=\rho}^{m} \binom{m}{i} \left(\frac{1}{\gamma!}\right)^i \left(1 - \frac{1}{\gamma!}\right)^{(m-i)}. \tag{20}$$

In the BRCA breast cancer data set, Ben-Dor et al. [4] report finding three statistically significant OPSMs, one with $\gamma = 4$ tissues (columns) and $\rho = 347$ genes (rows) with $U(4, 347) = 8.83 \cdot 10^{-51}$, another with 6 tissues and 42 genes with $U(6, 42) = 8.85 \cdot 10^{-19}$, and a third involving 8 tissues and 7 genes with $U(8, 7) = 0.0497$.

4.2.3 Solving Real Data Sets

The BRCA data set is considerably larger than our synthetic data sets. It is not surprising, then, that initial testing on this data using Algorithm **K** indicated that while CPLEX locates rather good feasible solutions, it experiences difficulty in closing the integrality gap from above. We believe this large gap was due to poor linear relaxations, resulting from both the

linearization of 0–1 variable products as well as incomplete information resulting from not using all maximal clique inequalities. In order to achieve faster convergence, it was therefore necessary to make some modifications to our solution approach.

Reducing the Feasible Region and Strengthening the Formulation. In order to facilitate a more rapid closing of the integrality gap, consider temporarily restricting the number of rows in an OPSM to be no more than a constant ρ. Similarly, let the number of columns be no more than γ, so that

$$\sum_i x_i \leq \rho, \tag{21}$$

$$\sum_j y_j \leq \gamma. \tag{22}$$

Then adding constraints (21) - (22) to any of the CF-OPSM variations, with well-chosen values of ρ and γ, will result in a formulation having a greatly reduced feasible region, one that CPLEX can much more easily handle. Furthermore, the optimal solution to this restricted problem forms a *valid* OPSM which is *feasible* for the original formulation without (21)-(22). Moreover, we can strengthen constraints (21) - (22) by appropriate multiplication of binary variables, a la the reformulation linearization technique (RLT) [23]. This yields the following additional valid inequalities:

$$\sum_i z_{ij} \leq \rho \cdot y_j \quad \forall\ j, \tag{23}$$

$$\sum_j z_{ij} \leq \gamma \cdot x_i \quad \forall\ i, \tag{24}$$

$$\gamma \sum_i x_i + \rho \sum_j y_j - \sum_i \sum_j z_{ij} \leq \gamma\rho, \tag{25}$$

$$\sum_i \sum_j z_{ij} \leq \gamma\rho, \tag{26}$$

where (23) and (24) are derived from (21) and (22) by multiplication of y_j and x_i, respectively, while (25) is derived by moving the right-hand sides of (21) and (22) to the left and multiplying.

The main idea in the solution approach of Ben-Dor et al. [4] is to first *restrict* the number of columns to a fixed value γ, and then search for OPSMs over the rows. With this in mind, by replacing (22), (24) and (25) with

$$\sum_j y_j = \gamma, \tag{27}$$

$$\sum_j z_{ij} = \gamma \cdot x_i \quad \forall\, i, \tag{28}$$

$$\gamma \sum_i x_i + \rho \sum_j y_j - \sum_i \sum_j z_{ij} = \gamma\rho, \tag{29}$$

respectively, we are also able to search for feasible OPSMs with fixed value γ.

This discussion then motivates an embedded algorithm that iteratively increases ρ and γ toward values m and n.

Embedded Algorithm with Partial OPSMs. For a given row h, define strictly increasing sequence $\{\rho\}_1^\pi = \{\rho_1, \ldots, \rho_\pi\}$ and nondecreasing sequence $\{\gamma\}_1^\pi = \{\gamma_1, \ldots, \gamma_\pi\}$, and let $\rho = \rho_1$, $\gamma = \gamma_1$. The optimal solution from Algorithm **K** augmented with constraints (21) - (22) and respective valid inequalities from (23)-(26) is a *partial* OPSM, *feasible* to the overall OPSM problem for row h. Now assume that both constraints (21) and (22) are non-binding for this optimal solution. Then it is not too difficult to show that we cannot obtain a larger OPSM for row h by increasing ρ and γ. On the other hand, if these constraints are tight, then by increasing the value of ρ to ρ_2 and γ to γ_2, we obtain a problem with a strictly larger feasible region, and can likely improve the previous iteration's solution. Furthermore, the partial OPSM from the previous iteration is also feasible for the larger region, and we can use it as a *warm start* to provide a valid lower bound. Iterating in this manner is a valid approach that converges to the largest OPSM for row h. Since the smaller formulations are much easier for CPLEX to handle, this strategy provides significant computational improvements.

The stopping condition of this approach, which is outlined in Algorithm 3, is either 1) when both constraints (21) and (22) are not binding, or 2) when we have used the final values in our sequence (i.e., $\rho = \rho_\pi$ and $\gamma = \gamma_\pi$). Note that it is not necessary to run this embedded algorithm over all m rows; for example, we can designate it to be run for only the first r rows. Then, for $r = 0$, Algorithm 3 is essentially Algorithm 2.

Algorithm 3 *[Embedded Algorithm]*

1. *Perform LP-based presorting of rows of A utilizing CF-OPSM formulation with nodal constraints (18) in the respective LP formulations. Assign* $h := 1$*, and read in sequences* $\{\rho\}$*,* $\{\gamma\}$*, and embedded algorithm iteration limit* r*.*
2. *Let embedded iteration counter* $\ell := 1$*, and set* $\rho = \rho_\ell$*,* $\gamma = \gamma_\ell$*.*
3. *Permutate columns of A such that the entries in row h occur in increasing order; call new matrix* $\hat{A}_h$*.*

4. *Generate formulation CF-OPSM for matrix $\hat{A}_h$ using nodal constraints (18).*

4(a). *Add additional constraint $x_h = 1$ and $x_k = 0$ for all $k < h$.*

4(b). *Generate maximal decreasing subsequences via modified FindLDS and FindLIS Algorithms (see discussion in Section 3.2.3); add respective valid inequalities (17).*

4(c). *If $h > \max\{1, r\}$, add a valid lower bound (19).*

4(d). *If $h > \max\{1, r\}$, solve LP relaxation of the IP formulation. Let $Z^*_{h_{LP}}$ be the obtained optimal objective function value. Go to 9 if $Z^*_{h_{LP}} \leq \bar{Z}$.*

4(e). *If $h \leq r$, add constraints (21)-(26) (or in the case of $\gamma_1 = \gamma_\pi$, constraints (21), (23), (26), (27)-(29)) to the formulation using current values of ρ and γ.*

5. *If $\ell > 1$ and $h \leq r$, read in initial feasible solution from warm start file.*

6. *Solve the obtained IP formulation. Let Z^h be the obtained optimal objective function value. If $h \leq r$, write out the optimal solution to a warm start file.*

7. *If $h > r$, go to 8. Otherwise $h \leq r$, so if $\ell = \pi$, OR if both constraints (21) and (22) are non-binding (or if constraint (22) is not present, just (21)), also go to 8. Else, let $\ell := \ell + 1$, set $\rho = \rho_\ell$, $\gamma = \gamma_\ell$, and go to 4.*

8. *Let $\bar{Z} = \max_{k=1,\ldots,h} Z^k$.*

9. *If $h < m$, let $h := h + 1$. Check **Stopping Criteria**, and if it is necessary to continue, go to 2 if $h \leq r$. Otherwise go to 3.*

10. *Return $Z^* = \bar{Z}$ and STOP.*

Depending on the sequences $\{\rho\}$, $\{\gamma\}$ and which constraints (22) or (27) are used in Algorithm 3, upon termination we can guarantee that we discover solutions that are either

(a) optimal, if $\rho_\pi = m$, $\gamma_\pi = n$ and formulation with (22) is used in Step **4(d)**, or

(b) optimal for a particular value of γ, if formulation with (27) is used in Step **4(d)**, or

(c) feasible and of high quality, corresponding to the values in the sequence $\{\gamma\}$.

We should also make note of the solver tuning we performed. Though there are only n column variables y_j in the CF-OPSM formulation ($n << m$), they are quite powerful, appearing in every composite product z_{ij}. Giving priority to (upward) branching in CPLEX

on this small subset of variables forces many y_j variables to 1, and doing so in conjunction with (22) or (27) serves to quickly eliminate portions of the branch-and-bound tree due to the maximal inversion inequalities (17) causing poor linear program relaxation values. Moreover, turning on CPLEX's barrier algorithm on both the root and child nodes of the branch-and-bound tree for a given row h helped to solve subproblems more quickly than the simplex algorithm. Lastly, CPLEX, through use of its proprietary tuning tool, recommended that we run the RINS heuristic every *50* iterations, set the CUTPASS parameter to *1*, and set the VARIABLESELECT (BRANCHING) parameter to *pseudo-reduced costs*. We implemented these recommendations.

In summary, Algorithm 3 and the associated solver tunings enables us to quickly find strong feasible solutions. Moreover, any feasible solutions the algorithm outputs in Step **6** are available for immediate data analysis purposes; it is not necessary to wait for convergence to the final solution. Lastly, for repeated runs on the same data set, Step **1** needs only be completed once; the presorted order of the rows can then be written to a file for subsequent runs. Indeed, we use this tactic to reduce our solution times in the following discussion.

4.2.4 Real Data: Results and Discussion

BRCA data. As we mentioned above for the BRCA data Ben-Dor et al. [5] reported statistically significantly patterns with $(\gamma, \rho) = (4, 347)$, $(\gamma, \rho) = (6, 42)$ and $(\gamma, \rho) = (8, 7)$. In order to compare our results, we used Algorithm 3 with the initial values of γ (fixed) and ρ set to match their largest dimensions.

Table 4: OPSMs in BRCA using Algorithm 3. Initial values of ρ (matching largest in Ben-Dor et al. [5]) in column 2; columns 3 and 4 display sequential results.

Cols γ	**Rows** ρ **(Time**† **in min)**		
4	**347** (220)	**520** (586)	**798** (1974)
6	**42** (71)	**63** (166)	**127** (2121)
8	**7** (17)	**10** (435)	**14** (2370)

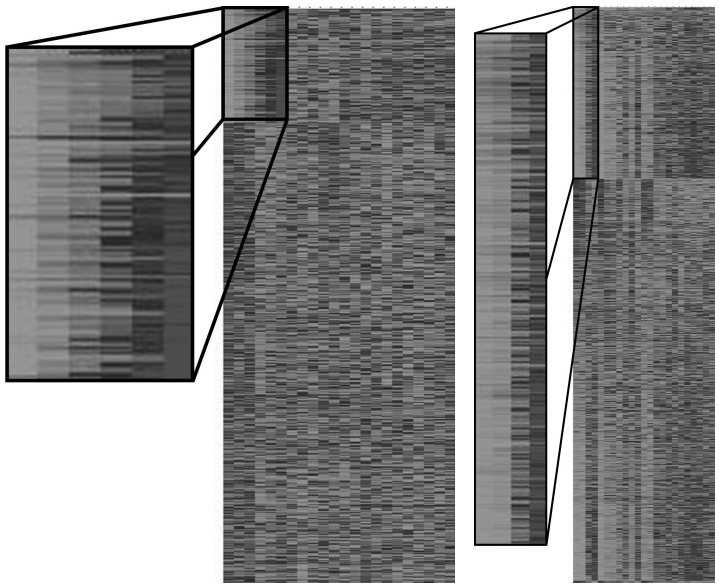
Knowing that our approach will eventually converge to the optimal solution, we present some feasible solutions from our results in Table 4, along with their solution times. For every case, we were able to locate OPSMs of larger size than were reported in Ben-Dor et al. [5], in reasonable amounts of time. It is worth noting here that the $(8, 14)$ OPSM we identified has $U(8, 14) = 5.88 \cdot 10^{-17}$, which significantly improves upon $U(8, 7) = 0.0497$, the result reported by Ben-Dor et al. [5]. Figure 3a displays an exemplary OPSM found using BRCA. Since it was performed only once (taking 623 minutes), the running time for Step **1**, LP-based presorting, is omitted from Table 4.

HuGE data. As in the previous experiment, we used Algorithm 3 with a sequence of fixed values for γ. Our results on this data set are displayed in Table 5. Included in the display are the statistical significance levels for the embedded OPSMs in HuGE. Figure 3b displays an exemplary OPSM found using HuGE. Again, as it was performed only once (taking 64 minutes), running time for Step **1**, LP-based presorting, has been omitted from Table 5.

Table 5: OPSMs in HuGE using Algorithm 3; statistical significance in final column.

Cols (γ)	**Rows** (ρ)	**Time**† **(min)**	$U(\gamma, \rho)$
3	569	335	$2.14 \cdot 10^{-146}$
4	335	489	$2.23 \cdot 10^{-176}$
5	180	1909	$1.84 \cdot 10^{-158}$
6	95	437	$7.43 \cdot 10^{-125}$
7	49	524	$6.98 \cdot 10^{-87}$
8	22	449	$8.87 \cdot 10^{-46}$
9	11	311	$1.79 \cdot 10^{-24}$

We feel these results represent favorable solution times for finding large OPSMs in HuGE. The statistical significance of these solutions indicate that they are of high quality. Only for fixed $\gamma = 5$ did it take more than one day to locate such large OPSMs; all other solutions were found in well under 12 hours.



(a) $(6, 127)$ in BRCA (truncated). (b) $(4, 335)$ in HuGE.

Figure 3: Two real data OPSMs using Algorithm 3.

5 Conclusions

In this paper we address exact solution methodologies for the Order Preserving Submatrix (OPSM) problem. We discuss theoretical computational complexity issues related to finding fixed patterns in matrices, and present exact approaches to solve the OPSM problem to optimality. The first is a general linear mixed 0–1 programming formulation, while the second is an iterative algorithm which makes use of a smaller linear 0–1 programming formulation of a restricted version of the initial problem. We discuss various algorithmic enhancements for the latter approach, which enable us to solve the OPSM problem to optimality for synthetic instances with approximately 1000

rows and 50 columns in a reasonable amount of time. Regarding real data, we discuss some additional enhancements to aid in the solution approach for these large data sets, along with the corresponding test results, which identified OPSMs larger than those reported in Ben-Dor et al. [4].

Some opportunities exist for the continuation of this study. For one, the criteria of rigid adherence to the strictly increasing pattern is rather inflexible. In the future, it will be interesting to consider relaxing the pattern criteria to locate approximate patterns – for instance, where a small minority of the rows do not satisfy the ordering criteria. Additionally, we can consider extensions of our formulation to handle the proposed w-OPSM problem.

Acknowledgments

We would like to thank three anonymous referees for their useful comments that greatly improved the quality of the paper. The research of the first author was supported by the US Department of Education Graduate Assistance in Areas of National Need (GAANN) Fellowship Program grant number P200A060149. The research of the second author was partially supported by NSF grant CMMI-0825993 and by US AFOSR grant number FA95500810268.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] B. Balasundaram, S. Butenko, and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005.

[3] M. Baz, B. Hunsaker, P. Brooks, and A. Gosavi. Automated tuning of optimization software parameters. Technical report, University of Pittsburgh Department of Industrial Engineering, 2007.

[4] 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 6th Annual International Conference on Computational Biology (RECOMB '02)*, pages 49–57. ACM, 2002.

[5] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. *Journal of Computational Biology*, 10(3-4):373–384, 2003.

[6] S. Busygin, O.A. Prokopyev, and P.M. Pardalos. Feature selection for consistent biclustering via fractional 0–1 programming. *Journal of Combinatorial Optimization*, 10(1): 7–21, 2005.

[7] S. Busygin, O.A. Prokopyev, and P.M. Pardalos. Biclustering in data mining. *Computers and Operations Research*, 35(9):2964–2987, 2008.

[8] Y. Cheng and G.M. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 93–103. AAAI Press, 2000.

[9] L. Cheung, D.W. Cheung, B. Kao, and K.Y. Yip. On mining micro-array data by order-preserving submatrix. *International Journal of Bioinformatics Research and Applications*, 3(1):42–64, 2007.

[10] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer Verlag, New York, 1999.

[11] R.J. Forrester and H.J. Greenberg. Quadratic binary programming models in computational biology. *Algorithmic Operations Research*, 3(2):110–129, 2008.

[12] H.K. Fung, S. Rao, C.A. Floudas, O.A. Prokopyev, P.M. Pardalos, and F. Rendl. Computational comparison studies of quadratic assignment like formulations for the in silico sequence selection problem in de novo protein design. *Journal of Combinatorial Optimization*, 10(1):41–60, 2005.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Company, 1979.

[14] I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, M. Raffeld, Z. Yahkini, A. Ben-Dor, E. Dougherty, J. Kononen, L. Bubendorf, W. Fehrle, S. Pittaluga, S. Gruvberger, N. Loman, O. Johannsson, H. Olsson, B. Wilfond, G. Sauter, O. Kallioniemi, A. Borg, and J. Trent.

Gene-expression profiles in hereditary breast cancer. *New England Journal of Medicine*, 344(8):539–548, 2001.

[15] HumanGene Expression Index. Huge index.org website. `http://www.hugeindex.org`, 2009.

[16] ILOG. *CPLEX 11.0 User's Manual*. ILOG CPLEX Division, Incline Village, NV, 2007.

[17] S.C. Madeira and A.L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE Transactions on computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[18] C.N. Meneses, Z. Lu, C.A.S. Oliveira, and P.M. Pardalos. Optimal solutions for the closest-string problem via integer programming. *INFORMS Journal on Computing*, 16 (4):419–429, 2004.

[19] A.T. Murray and R.L. Church. Constructing and selecting adjacency constraints. *INFOR*, 34(3):232–248, 1996.

[20] A.T. Murray and R.L. Church. Facets for node packing. *European Journal of Operational Research*, 101(3):598–608, 1997.

[21] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, 1973.

[22] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13(2):179–191, 1961.

[23] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, 1998.

[24] M.P. Tan, J.R. Broach, and C.A. Floudas. A novel clustering approach and prediction of optimal number of clusters: Global optimum search with enhanced positioning. *Journal of Global Optimization*, 39(3):323–346, 2007.

[25] A. Trapp, O.A. Prokopyev, and S. Busygin. Finding checkerboard patterns via fractional 0–1 programming. *Journal of Combinatorial Optimization*, pages 1–26, 2009.