Approximation Algorithms for a Minimization Variant of the Order-Preserving Submatrices and for Biclustering Problems

DORIT S. HOCHBAUM, University of California, Berkeley
ASAF LEVIN, The Technion

Finding a largest Order-Preserving SubMatrix, OPSM, is an important problem arising in the discovery of patterns in gene expression. Ben-Dor et al. formulated the problem in Ben-Dor et al. [2003]. They further showed that the problem is NP-complete and provided a greedy heuristic for the problem. The complement of the OPSM problem, called MinOPSM, is to delete the least number of entries in the matrix so that the remaining submatrix is order preserving. We devise a 5-approximation algorithm for the MinOPSM based on a formulation of the problem as a quadratic, nonseparable set cover problem. An alternative formulation combined with a primal-dual algorithm improves the approximation factor to 3. The complexity of both algorithms for a matrix of size $m \times n$ is $O(m^2 n)$. We further comment on the related biclustering problem.

Categories and Subject Descriptors: G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Gene expression, order preserving, approximation algorithm, biclustering

ACM Reference Format:
Hochbaum, D. S. and Levin A. 2013. Approximation algorithms for a minimization variant of the order-preserving submatrices and for biclustering problems. *ACM Trans. Algorithms* 9, 2, Article 19 (March 2013), 12 pages.
DOI: http://dx.doi.org/10.1145/2438645.2438651

1. INTRODUCTION

According to Cheng and Church [2000]: "In expression data analysis, the uttermost important goal . . . is the finding of a set of genes showing strikingly similar up-regulation and down-regulation under a set of conditions."

The discovery of patterns in gene expression matrices is abstracted as the identification of an *order-preserving submatrix*. The input to the problem is an $m \times n$ real gene expression matrix $A = (a_{ij})$. The entries in columns represent different gene expressions during a particular experiment or subject to a specific condition. A pattern is a $p \times q$ submatrix $B = (b_{ij})$ of A so that all rows increase or decrease uniformly. That is, for $j_1 \neq j_2$ and for all $i = 1, \ldots p$, $b_{ij_1} < b_{ij_2}$, or for all $i = 1, \ldots p$, $b_{ij_1} > b_{ij_2}$. This is equivalent to having a permutation of the q columns of B such that all rows have (strictly) increasing elements: for $j_1 < j_2$ $b_{ij_1} < b_{ij_2}$ for all $i = 1, \ldots p$. The pattern

The research of D. S. Hochbaum was supported in part by NSF award nos. CMMI-1200592 and CBET-0736232.
Authors' addresses: D. S. Hochbaum, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720; A. Levin (corresponding author), Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel; email: levinas@ie.technion.ac.il.

DOI: http://dx.doi.org/10.1145/2438645.2438651

expressed in the submatrix B might arise due to similar responses to different stages of an experiment, or different conditions, representing similar response over time, or representing distinct stages in the progress of a disease or in a cellular process. The objective is to identify such submatrix B that has the largest number of elements in it. The submatrix B is called *order preserving*.

The Order-Preserving SubMatrix (OPSM) problem is to identify a submatrix of A that is order preserving so that the total number of entries in the submatrix is maximized. The problem is then to find a subset of columns of the given matrix A and a permutation on these columns, and a subset of rows so that for the permutation on the subset of columns all rows are monotone increasing.

Ben-Dor et al. [2003] formulated the OPSM problem and showed that in the general case it is NP-hard. They proposed a greedy heuristic algorithm for finding a hidden order-preserving submatrix.

For the OPSM problem we differentiate between two parts of the problem. One part is to determine the permutation of the columns in the order-preserving submatrix. The second part is to find a largest order-preserving submatrix that consists of a subset of rows that conform to the determined permutation restricted to a subset of columns selected. For the first part, for a data matrix $A_{m\times n}$ the number of column permutations is no more than $n!$. This number can be reduced considerably: The number of permutations may not exceed the number of rows m as we are only interested in permutations that agree with at least one row. Therefore the approximation algorithm will be applied for one row permutation at a time and select the largest OPSM found. We further observe that it is possible to replace the matrix A by n copies of itself, $\mathcal{A} = [A, A, \ldots, A]$. An order-preserving submatrix of $\mathcal{A}$ never contains two copies of the same column of A as the entries are of equal value, instead of strictly increasing. Thus any OPSM of $\mathcal{A}$ corresponds to an OPSM of A. In that case one can apply an optimal algorithm to the matrix $\mathcal{A}$ with the identity permutation.

The search for the maximum order-preserving submatrix is equivalent to the complementary problem of minimizing the number of entries in the matrix that are *deleted* from the matrix A. This equivalence is with respect to the optimal solutions to both problems, and does not carry to approximation algorithms' equivalence. So the reduction between the two problems is not an approximation-preserving reduction. It is this minimization problem named MinOPSM that we study here, and for which we provide approximation algorithms. As demonstrated here, the formulation of the problem affects greatly its solvability and the approximation algorithms that can be derived. Our main contribution in this article is a 3-approximation algorithm for the MinOPSM problem of complexity $O(m^2n)$. Our algorithm is based on the primal-dual scheme using a formulation of the problem with $O(mn^2)$ constraints for a given permutation of the columns. We identify special properties of the problem that allow the algorithm to consider only $O(mn)$ constraints. This compact representation of the constraints used throughout the algorithm contributes to small storage space and is the key-point in our improved time complexity. Our approximation algorithms are the first known approximation algorithms for the MinOPSM problem.

Note that the transformation of A with arbitrary permutation to $\mathcal{A}$ with the identity permutation preserves the value of an optimal solutions to both the OPSM problem and to the MinOPSM problem. However, when we apply approximation algorithms, the transformation preserves the approximation ratio when applied to OPSM, but it does not preserve the approximation ratio when applied to MinOPSM. Thus, our approximation algorithms for MinOPSM which use a fixed permutation will be applied m times, once for each row.

As we are approximating MinOPSM and not the OPSM problem itself, our results do not imply a constant approximation algorithm for OPSM.

The OPSM and MinOPSM problems belong to a family of *biclustering* problems. Biclustering was defined by Mirkin [1996] as the simultaneous clustering of both row and column sets in a "data matrix". One common specific biclustering problem refers to a $0-1$ matrix where the object is to find a largest submatrix that has all entries equal 1. As we will see, the definition of the "size" of the submatrix affects the complexity of the problem. We prove that one of these biclustering problems is related to a special case of MinOPSM. The biclustering problems are observed to be node or edge deletion in a bipartite graph so as to obtain a biclique (a complete bipartite graph). The relevance of the results on approximating problems related to bicliques, in Hochbaum [1982], is pointed out.

Article Outline. We describe in Section 2 our 5-approximation algorithm, and present its analysis. This approximation algorithm serves to present the main ideas that are used in Section 3 to obtain our improved 3-approximation algorithm for MinOPSM. In Section 4 we consider the biclustering problem. Finally, Section 5 provides brief concluding remarks.

2. A FORMULATION OF MINOPSM AND A 5-APPROXIMATION ALGORITHM

2.1. MinOPSM Problem Formulation

As discussed in the Introduction, the MinOPSM problem is reducible to the problem solved for one permutation: the identity permutation. The solution is taken to be the best (giving a largest OPSM) among the m permutations corresponding to the m rows of A.

Let r_i be a binary variable that is equal to 1 if the corresponding row i is excluded from the submatrix and c_j is a binary variable equal to 1 if the corresponding column j is excluded from the submatrix.

For each row i that has a pair of entries violating the monotone increasing order, $j < k$ and $a_{ij} \geq a_{ik}$, both a_{ij} and a_{ik} cannot be present in the order-preserving submatrix. There are three possibilities: either row i is excluded from the order-preserving submatrix, or column j is excluded, or column k is excluded. The following optimization problem is then equivalent to MinOPSM.

$$\begin{array}{lll} \text{(OPSM1)} & \min & \sum_{i=1}^{m} nr_i + \sum_{j=1}^{n} mc_j - (\sum_{i=1}^{m} r_i)(\sum_{j=1}^{n} c_j) \\ \text{subject to} & & r_i + c_j + c_k \geq 1 \quad \forall i \text{ and for all} j < k \text{ such that } a_{ij} \geq a_{ik} \\ & & r_i, c_j \text{ binary} \quad \forall i, j. \end{array}$$

In this formulation the objective function is the same as that of MinOPSM. To see that, observe that each deleted entry is counted exactly once in the objective function of OPSM1. This is because a deleted entry is counted exactly once in the first two terms if either its row or its column are deleted but not both. If both the column and row of a deleted entry are deleted then it is counted twice in the first two terms. But then the number of such entries is counted in the third term of the objective function and subtracted from the total.

It is evident from this formulation that this problem is a set cover problem (the elements of the set cover instance are the constraints of OPSM1, and the sets are the variables of this formulation where a set contains all constraints that will be satisfied if we set the corresponding variable to 1), albeit with an objective that contains a nonlinear term. The constraint matrix of this problem is that of a set cover where each element can be covered by three sets at most. In other words, each row has at most three 1s in it. For such a problem, with each row sum of the constraint matrix not exceeding 3, the algorithm of Hochbaum [1982] gives a factor-3 approximation. This is, however, for a *linear objective* set cover problem. So we can 3-approximate the

problem with the linear part of the objective $\min \sum_{i=1}^{m} nr_i + \sum_{j=1}^{n} mc_j$. Let the number of rows and columns in the 3-approximate solution to the linear set cover be R^A, C^A.

2.2. The Approximation Factor

Here we adjust the approximation factor of the set cover problem to account for the nonlinear objective. The question we address is how different the approximation for the objective of $\min \sum_{i=1}^{m} nr_i + \sum_{j=1}^{n} mc_j$ is from the nonlinear objective?

Firstly $\sum_{i=1}^{m} nr_i + \sum_{j=1}^{n} mc_j > 2(\sum_{i=1}^{m} r_i)(\sum_{j=1}^{n} c_j)$. So this immediately yields a 6-approximation algorithm. With a tighter analysis we show next that this is a 5-approximation algorithm.

Recall that $R^A = \sum_{i=1}^{m} r_i$ and $C^A = \sum_{j=1}^{n} c_j$ where r_i and c_j are according to the solution obtained by the algorithm. Therefore, the solution derived by the algorithm, R^A, C^A, corresponds to the (nonlinear) objective value $a^A - c^A = nR^A + mC^A - R^A C^A$ where $a^A = nR^A + mC^A$ and $c^A = R^A C^A$. Consider the optimal solution (r^*, c^*) to the linear programming relaxation of (OPSM1) with the linear objective function. Denote $R^O = \sum_{i=1}^{m} r_i^*$ and $C^O = \sum_{j=1}^{n} c_j^*$. Thus the optimal solution value is $a^O - c^O = nR^O + mC^O - R^O C^O$ where $a^O = nR^O + mC^O$ and $c^O = R^O C^O$. Then,

$$\frac{a^A - c^A}{a^O - c^O} = \frac{a^A}{a^O} + \frac{a^A c^O - a^O c^A}{a^O(a^O - c^O)} \leq \frac{a^A}{a^O} + \frac{(a^A - a^O)c^O}{a^O(a^O - c^O)},$$

where the equality holds by simple arithmetic and the inequality holds because $c^A \geq c^O$. This last inequality holds because the approximate solution is obtained by rounding up the components of (r^*, c^*). Now $a^A \leq 3a^O$ follows from the performance guarantee of the 3-approximation and therefore we conclude

$$\frac{a^A - c^A}{a^O - c^O} \leq 3 + \frac{2a^O c^O}{a^O(a^O - c^O)} = 3 + \frac{2c^O}{a^O - c^O}.$$

Furthermore, $a^O \geq 2c^O$. So,

$$\frac{a^A - c^A}{a^O - c^O} \leq 3 + \frac{2c^O}{c^O} = 5.$$

This discussion has thus established the following.

LEMMA 2.1. *The set cover approximation algorithm is a* 5*-approximation algorithm for MinOPSM.*

2.3. The Complexity

The algorithm of Bar-Yehuda and Even [1981] improves the complexity of the algorithm of Hochbaum [1982] for the set cover problem. That algorithm selects one uncovered element, or unsatisfied constraint, at a time, and a variable, or set, among the candidates that can cover that element. The variable chosen in the cover is the variable of minimum reduced cost. The algorithm then updates the reduced costs of other sets that cover the same element so as to maintain dual feasibility (see Hochbaum [1996] for details). The complexity of this approximation algorithm is linear in the number of 1s in the constraint matrix. Since each row in the constraint matrix has three 1s this complexity is 3 times the number of constraints. We observe, however, that for each row of the matrix there are at most $O(n^2)$ constraints for a total of $O(mn^2)$ constraints.

Although the formulation OPSM1 has $O(mn^2)$ constraints, we demonstrate that the algorithm's complexity is only $O(mn)$. This is achieved by generating only $O(mn)$ constraints throughout the execution of the algorithm.

We note first that when a variable or a set, in set cover terminology, is selected in the cover, it then covers and thus eliminates either all constraints corresponding to a column or all constraints corresponding to a row depending on whether a column or row variable were selected. Therefore the number of variable selection steps in the algorithm is only $O(m+n)$ and thus linear in the dimension of the matrix A. We next address the complexity of each step. To simplify notation we define a *conflict set* for each row i, $\mathcal{C}_i$, as the collection of ordered pairs $j_1 < j_2$ such that $a_{ij_1} \geq a_{ij_2}$.

At each step of the algorithm, a violated constraint that is not satisfied by the current solution needs to be identified and then either one column or one row is deleted. In order to find a violated constraint, we maintain a queue $\mathcal{Q}$ of a subcollection of the violated constraints, a linked list $\mathcal{C}$ of nondeleted columns, and a linked list $\mathcal{R}$ of nondeleted rows (see Chapter 10 in Cormen et al. [2001] for definitions of these data structures). Two columns j and k, for $j < k$, are said to be *adjacent in* $\mathcal{C}$ if $j, k \in \mathcal{C}$ and for all columns t such that $j < t < k$, $t \notin \mathcal{C}$ – namely, t has already been deleted. If j and k are adjacent in $\mathcal{C}$ and $j < k$, we set $k = next(j)$ in the linked list $\mathcal{C}$. Throughout the algorithm a violated constraint $r_i + c_j + c_k \geq 1$ is added to $\mathcal{Q}$ only if j and k are adjacent in $\mathcal{C}$ and $i \in \mathcal{R}$. In spite of this apparent restriction, we prove that if there is a violated constraint, then there is at least one such constraint in $\mathcal{Q}$.

LEMMA 2.2. *If the set of violated constraints is nonempty, then there exists a violated constraint $r_i + c_j + c_k \geq 1$ such that j and k are adjacent in $\mathcal{C}$ and $i \in \mathcal{R}$.*

PROOF. Suppose by contradiction that the claim does not hold. Then each row is monotone increasing along the sequence of nondeleted adjacent columns, as otherwise there will be a constraint with adjacent columns. Hence the nondeleted entries form an order-preserving submatrix and there are no violated constraints. This contradicts the assumption that the set of violated constraints is nonempty. □

We next describe the initialization of the data structures. $\mathcal{R}$ is initialized as the list of all m rows, and similarly $\mathcal{C}$ is initialized as the list of all n columns where for all $j = 1, 2, \ldots, n-1$ column $j+1$ is the successor of column j in $\mathcal{C}$ (denoted as $j+1 = next(j)$). In all iterations we maintain $\mathcal{C}$ as a sorted list of indices of the nondeleted columns. To initialize $\mathcal{Q}$ we need to traverse each row i of the matrix, and check for all values of $j = 1, 2, \ldots, n-1$ if $(j, j+1) \in \mathcal{C}_i$ (and if so add the constraint $r_i + c_j + c_{j+1} \geq 1$ to $\mathcal{Q}$). So $\mathcal{Q}$ is initialized to the following set of constraints $\{r_i + c_j + c_{j+1} \geq 1 : i = 1, 2, \ldots, m; (j, j+1) \in \mathcal{C}_i\}$. This initialization of $\mathcal{Q}$ takes $O(mn)$ time because we need to check only $O(n)$ pairs $(j, j+1)$ for all values of i. Hence, the initialization of our data structures takes $O(mn)$ time.

Each iteration begins by popping a constraint from $\mathcal{Q}$. If this constraint is no longer violated then it is discarded and the next constraint is popped from $\mathcal{Q}$. If it is a violated constraint we apply the iteration of the algorithm of Bar-Yehuda and Even [1981]. At the end of the iteration, the algorithm determines whether to delete a row or a column. If the algorithm deletes a column j, then denote by t the column immediately preceding j in $\mathcal{C}$ prior to this deletion (so $j = next(t)$). Denote by k the successor of j in $\mathcal{C}$ prior to the deletion of j from $\mathcal{C}$ (so $k = next(j)$). The algorithm then scans the undeleted rows of the matrix and checks, for each row $s \in \mathcal{R}$, whether the following constraint is satisfied: $a_{s,t} < a_{s,k}$. If it is not satisfied, then the (violated) constraint $r_s + c_t + c_k \geq 1$ is pushed to the queue $\mathcal{Q}$. Therefore, when a column is deleted, there are at most $O(m)$ possible constraints to consider and check whether any of these constraints have to be pushed into $\mathcal{Q}$. We conclude that each iteration resulting in a deletion of a column incurs an additional running time of $O(m)$, whereas each iteration resulting in a deletion of a row takes $O(1)$ time. For this analysis of the running time, we charge the time of popping

a constraint once it is no longer violated, to the iteration in which it was inserted to $\mathcal{Q}$. Therefore, the total running time of the resulting algorithm is $O(mn)$.

At the end of the algorithm $\mathcal{Q}$ is empty, and hence, from Lemma 2.2, the resulting submatrix is order preserving. Since this approximation algorithm is repeated for each row, ordering the matrix according to a monotone increasing ordering of the row entries, the running time is $O(m^2n)$.

3. A PARTIAL COVER FORMULATION AND A 3-APPROXIMATION ALGORITHM

Here we consider an alternative formulation of the MinOPSM problem. This formulation is conditioned on the number of rows deleted not exceeding (or rather equal to) a given constant k while minimizing the number of columns deleted.

$$\begin{array}{lll} \text{(OPSM2)} & \min & \sum_{j=1}^{n} c_j \\ \text{subject to} & & r_i + c_{j_1} + c_{j_2} \geq 1 \quad \forall i \text{ and } \forall j_1 < j_2 \text{ such that } a_{ij_1} \geq a_{ij_2} \\ & & \sum_{i=1}^{n} r_i \leq k \\ & & r_i, c_j \text{ binary} \quad \forall i, j \end{array}$$

The requirement that the variables are binary can be replaced with nonnegativity and integrality requirement. The upper bound of 1 on the values of the variables will be satisfied in any optimal solution.

We observe that OPSM2 is a generalization of the partial vertex k-cover problem. The latter problem is to find a vertex cover for at least $|E| - k$ edges of a graph with the least number (weight) of vertices. In the partial vertex k-cover problem's formulation there is a constraint for each edge $e_i = [j_1, j_2]$, $r_{e_i} + c_{j_1} + c_{j_2} \geq 1$. Thus in this problem r_i appears only once, in the one constraint that has j_1 and j_2 only. So in the partial vertex cover problem $r_i = 1$ means that i is not covered and $c_{j_1} = c_{j_2} = 0$. In contrast, in OPSM2, r_i is associated with a *set* of constraints, and $r_i = 1$ if not all of them are satisfied.

We now formulate the dual to the OPSM2 problem. Let the dual variables be $u_{ij_1j_2}$ and z the dual variable of the constraint bounding the number of deleted rows.

$$\begin{array}{lll} \text{(Dual-OPSM)} & \max & \sum_{i=1}^{m} \sum_{(j_1, j_2) \in \mathcal{C}_i} u_{ij_1j_2} - kz \\ \text{subject to} & & \sum_{(j_1, j_2) \in \mathcal{C}_i} u_{ij_1j_2} \leq z, \;\; i = 1, \ldots, m \\ & & \sum_{i=1}^{m} \left(\sum_{(j, j_2) \in \mathcal{C}_i} u_{ijj_2} + \sum_{(j_1, j) \in \mathcal{C}_i} u_{ij_1j} \right) \leq 1 \;\; j = 1, \ldots, n \\ & & u_{ij_1j_2} \geq 0 \;\; \forall i \text{ and } (j_1, j_2) \in \mathcal{C}_i \\ & & z \geq 0 \end{array}$$

We will maintain implicitly $z = \max_i \sum_{(j_1, j_2) \in \mathcal{C}_i} u_{ij_1j_2}$. The approximation algorithm we use here is a primal-dual algorithm.

3.1. Data Structures

Our algorithm uses the following data structures and maintains them in each iteration.

The indices of the nondeleted columns are maintained as a linked list L. These indices are sorted in monotone increasing order. For a column $j \in L$ we denote by $next(j)$ the successor of j in L (that is, the minimum index k of a column in L such that $k > j$). As in Section 2.3 j_1 and j_2 are said to be adjacent in L (for $j_1 < j_2$) if $j_1, j_2 \in L$ and $j_2 = next(j_1)$. The algorithm makes use of the fact established in Lemma 2.2 that if there is a violated constraint then there is one that involves adjacent column indices in L.

We note that whereas the sets $\mathcal{C}_i$ are fixed, these sets may include too many pairs (at most $O(n^2)$ pairs in each such set). To reduce the resulting time complexity, we maintain small subsets of these sets denoted as $\mathcal{C}_i(L) \subseteq \mathcal{C}_i$ (these sets are dynamic and hence changed along the algorithm). For each row i, we let $\mathcal{C}_i(L)$ be the set of all pairs (j_1, j_2) in $\mathcal{C}_i$ such that j_1 and j_2 are adjacent in L. We assume that $\mathcal{C}_i(L)$ is an ordered list of pairs (with arbitrary order). The main properties that we will use are derived from Lemma 2.2: (1) $\mathcal{C}_i(L) = \emptyset$ if and only if there are no $j, k \in L$ such that $j < k$ and $a_{i,j} \geq a_{i,k}$, and (2) throughout the algorithm $|\mathcal{C}_i(L)| \leq n$.

For each column j we maintain the total sum of the dual variables corresponding to j, U_j, where

$$U_j = \sum_{i=1}^{m} \left(\sum_{j_1=1,\ldots,\, j-1:(j_1,\, j)\in\mathcal{C}_i} u_{ij_1j} + \sum_{j_2=j+1,\ldots,n:(j,\, j_2)\in\mathcal{C}_i} u_{ijj_2} \right).$$

For each column j we further maintain a list of all positions in which j appears in $\mathcal{C}_i(L)$, and a pointer from the column to its position in L.

3.2. The Algorithm

The algorithm works as follows: A dual feasible solution is initialized by setting $u_{ij_1j_2} = 0$ for all i and all $(j_1, j_2) \in \mathcal{C}_i$, and $z = 0$. There is also a trivial (and infeasible) primal solution consisting of the empty set of columns. To initialize the data structures we let L consist of the indices $1, 2, \ldots, n$ where for all j, $j + 1 = next(j)$. For all j, U_j is initialized to zero. We also initialized $\mathcal{C}_i(L)$ and the list of all positions in which j appears in $\mathcal{C}_i(L)$ by traversing the input matrix once.

Each iteration of the primal-dual algorithm consists of increasing (simultaneously) the dual variables $u_{ij_1j_2}$ for all i such that $\mathcal{C}_i(L) \neq \emptyset$ and for (j_1, j_2) that is the first pair in the list $\mathcal{C}_i(L)$. The increase is at a unit rate, and therefore z is also increased at a unit rate. The dual variables are increased until at least one constraint for j, $\sum_{i=1}^{n} \left(\sum_{(j,j_2)\in\mathcal{C}_i} u_{ijj_2} + \sum_{(j_1,j)\in\mathcal{C}_i} u_{ij_1j} \right) \leq 1$ becomes tight. Then j is added to the set of selected columns, and the list L and the sets $\mathcal{C}_i(L)$ are updated for all i.

The algorithm halts the first time when there are at least $m - 3k$ row indices i such that $\mathcal{C}_i(L) \neq \emptyset$. Then, we delete the rows i such that $\mathcal{C}_i(L) = \emptyset$, and the columns that were deleted along the way (i.e., the columns not in L). Since for the nondeleted rows $\mathcal{C}_i(L) = \emptyset$, we conclude that for a nondeleted row i there are no $j, k \in L$ such that $(j, k) \in \mathcal{C}_i$.

3.3. A Fast Implementation

To find the selected column j it is necessary to determine which U_j becomes first equal to one. Such j must obviously be a column where at least one of its dual variables is increased during the current iteration. For each row i with $\mathcal{C}_i(L) \neq \emptyset$ we mark the two columns j_1, j_2 that form the first pair, such that $u_{ij_1j_2}$ is raised in the current iteration. Thus U_j is raised at a rate equal to the number of marks the j-th column has. We can hence find in $O(m)$ time the value of j which is the new column selected in the solution, as for each j we can compute the time point in which U_j will equal one (assuming it increases in the same rate as defined for the current iteration). If there are several columns j such that $U_j = 1$ simultaneously, then we arbitrarily pick one such column. The other candidate columns will be tight in the next iterations assuming that there will be increasing dual variable corresponding to them.

We claim that the update of $U_{j'}$ is done in $O(m)$, per selection iteration, for all values of j'. To see that, observe that it is necessary to update only the values of $U_{j'}$ such that

column j' is marked (at least once). Therefore, there are only $O(m)$ update operations to apply (at most two column indices per row of the matrix). Each such update operation takes a constant time, because we know the number of marks of column j', and $U_{j'}$ has been increased (in the current iteration) at a rate equal to the number of marks that column j' has.

To complete the implementation details, consider the update of L (arising by the deletion of column j) and the update of $\mathcal{C}_i(L)$ for all i. To update L, we find the location of j in L in constant time using the columns pointers. If, prior to the deletion of j, the predecessor of j in L was j_1, and the successor of j in L is j_2 (so $next(j_1) = j$ and $next(j) = j_2$), then L is updated by skipping over j and deleting the column pointer of column j. This is done by setting $next(j_1) = j_2$. Thus the update of L is carried in constant time. To implement the updates of $\mathcal{C}_i(L)$ for all i, note that the only pair of columns that become adjacent in L are (j_1, j_2), for $next(j_1) = j$ and $next(j) = j_2$. So for each i, we check if $(j_1, j_2) \in \mathcal{C}_i$. This involves checking whether $a_{ij_1} \geq a_{ij_2}$, which is done in constant time for a fixed value of i. If so, we add (j_1, j_2) to $\mathcal{C}_i(L)$. We also need to delete all positions in which j appears in $\mathcal{C}_i(L)$. This last step is performed in $O(m)$ time using the list maintained for column j. We conclude that each iteration takes $O(m)$ time.

THEOREM 3.1. *The total running time of the algorithm for all values of k and a fixed permutation of the columns is $O(mn)$.*

PROOF. The running time of the algorithm (for a given value of k) is $O(mn)$ because the number of iterations is at most the number of columns (as each iteration deletes at least one column of the matrix). We next note that when we decrease k by 1 (i.e., $k' = k - 1$) we do not need to restart the algorithm from scratch, as we can continue the algorithm until there are at least $m - 3k'$ row indices such that $\mathcal{C}_i(L) = \emptyset$. Hence, the total running time of the algorithm for all values of k is also $O(mn)$. □

It remains to analyze the performance guarantee of the algorithm.

3.4. Approximation Ratio

The rate at which the dual variables are increased ensures that for each i such that $\mathcal{C}_i(L) \neq \emptyset$, $\sum_{(j_1,j_2)\in\mathcal{C}_i} u_{ij_1j_2} = z$. This is so since both sides are increased at a unit rate in each iteration as long as $\mathcal{C}_i(L) \neq \emptyset$. (Note that in the left-hand side there is at most one term that is increased in a fixed iteration.)

Consider the iteration of the algorithm where k equals the number of deleted rows in a fixed optimal solution. We note that the number of deleted rows is at most $3k$. We next show that the number of selected columns is at most three times the optimal solution to MinOPSM.

THEOREM 3.2. *The number of selected columns by the primal-dual algorithm is at most 3 times the optimal number of selected columns in the optimal solution to OPSM2.*

PROOF. We allocate the dual cost to *pay* for the primal solution as follows. First, consider the set S_k of the $3k$ last rows that have nonempty $\mathcal{C}_i(L)$ (breaking ties arbitrarily). For $i \in S_k$, for all $(j_1, j_2) \in \mathcal{C}_i$ we allocate the dual cost of $u_{ij_1j_2}$ at most three times as follows: once to pay for the row i, once to pay for column j_1, and once to pay for column j_2. For $i \notin S_k$, and for every $(j_1, j_2) \in \mathcal{C}_i$ the dual variable $u_{ij_1j_2}$ is used at most twice: once to pay for column j_1 and once to pay for column j_2. Note that for $i \in S_k$ the total payments for row i are exactly z (as $z = \sum_{(j_1,j_2)\in\mathcal{C}_i} u_{ij_1j_2}$ throughout

the algorithm because we increase both sides at the same rate), and hence the total payment of all the rows is exactly $3kz$. Therefore,

$$\begin{aligned}(Dual - OPSM) &= \frac{1}{3} \cdot \left(3 \sum_i \sum_{(j_1,j_2) \in \mathcal{C}_i} u_{ij_1j_2} - 3kz \right) \\ &\geq \frac{1}{3} \cdot \left[2 \sum_i \sum_{(j_1,j_2) \in \mathcal{C}_i} u_{ij_1j_2} + \left(\sum_{i \in S_k} \sum_{(j_1,j_2) \in \mathcal{C}_i} u_{ij_1j_2} - 3kz \right) \right] \\ &= \frac{1}{3} \cdot 2 \sum_i \sum_{(j_1,j_2) \in \mathcal{C}_i} u_{ij_1j_2}.\end{aligned}$$

A deleted column receives payments of at least one, and therefore the number of columns that the algorithm deletes is at most the total payments for the columns, and hence it is at most $2 \sum_i \sum_{(j_1,j_2) \in \mathcal{C}_i} u_{ij_1j_2}$. By the preceding inequality the total cost of the primal solution is at most three times the (Dual-OPSM). □

Therefore, our primal-dual algorithm provides the same performance guarantee as the earlier algorithm (based on solving the linear program relaxation for each value of R, C). Therefore, we establish the following.

THEOREM 3.3. *There is a 3-approximation algorithm for MinOPSM and identity permutation with running time of $O(mn)$.*

PROOF. Assume that the optimal solution deletes R rows and C columns. Then in the iteration of the algorithm where k=R the algorithm will delete $3R$ rows and (at most) $3C$ columns. The number of entries that the optimal solution deletes is $nR+(m-R)C$ whereas the algorithm deletes at most $3nR + (m - 3R)3C = 3nR + 3mC - 9RC \leq 3nR + 3mC - 3RC = 3 \cdot (nR + (m - R)C) = 3OPT$. And hence the algorithm is a 3-approximation algorithm for the minimization version. □

Using the argument given in the Introduction we conclude that there is a 3 - approximation algorithm for MinOPSM the running time of which is $O(m^2n)$.

The WEIGHTED MINOPSM problem is similar to MinOPSM except that each entry of A has a nonnegative weight and the goal is to minimize the total weight of deleted entries. We claim that our algorithm for MinOPSM can be extended to weighted MinOPSM with the same approximation factor. To see this define a weight of a row (column) to be the total weight of the entries in the row (column). Next apply the algorithm for MinOPSM where a column j becomes tight when the U_j equals its weight, and run the algorithm until the total weight of the rows with nonempty $\mathcal{C}_i(L)$ becomes at most $3k$ (for the first time) and these rows are deleted. When k is increased we need not restart the algorithm but rather continue to get the remainder of the candidate solutions. In total there are at most n solutions that we consider, and using the same analysis as in the unweighted case we get a 3-approximation algorithm. We have thus established the following proposition.

PROPOSITION 3.4. *There is a 3-approximation algorithm for the weighted MinOPSM problem that runs in $O(m^2n)$ time.*

4. THE BICLUSTERING PROBLEM

The biclustering problem is defined on a $0 - 1$ input matrix A. The problem we call here *biclustering* is to find a subset of columns and rows forming a largest submatrix

Table I.

OBJECTIVE FOR SUBMATRIX	COMPLEXITY	APPROXIMATION FACTOR (for the complementary minimization problem)
Maximum Weighted number of rows and columns in all 1s submatrix	Polynomial time	
Maximum number of 1 entries	NP-hard	2
Maximum weighted number of entries	NP-hard	2
Maximum number of entries and balanced	NP-hard	unknown
Maximum number of entries and Order-preserving (weighted)	NP-hard	3

The complexity of biclustering according to the objective and approximation factors for the complementary minimization problem.

consisting entirely of 1 valued entries. The sense in which the submatrix is largest impacts the complexity of the problem, as we see in Table I.

We first prove that the biclustering problem is at least as difficult as a special case of the MinOPSM problem. We define the concept of an *almost-order-preserving* matrix. A is almost-order preserving if for every $i = 1, 2, \ldots, m$ and every $j = 1, 2, \ldots, n-2$ and all $k = 2, 3, \ldots, n-j$, $a_{i,j} < a_{i,j+k}$. To see that any order-preserving matrix is also almost-order preserving note that we can define a matrix to be order preserving if for every $i = 1, 2, \ldots, m$ and every $j = 1, 2, \ldots, n-2$ and all $k = 1, 2, 3, \ldots, n-j$, $a_{i,j} < a_{i,j+k}$. So the difference is that the constraints for the value $k = 1$ are present, for order-preserving matrices, as well.

LEMMA 4.1. *The bilclustering problem is at least as difficult as the special case of MinOPSM where the input matrix is almost-order preserving.*

PROOF. Given an input to the MinOPSM problem defined on an input matrix A which is almost-order preserving. For such instances we construct a matrix $B_{m\times(n-1)}$ such that $B_{ij} = 1$ if $a_{i,j} < a_{i,j+1}$ and otherwise $B_{i,j} = 0$. Then, we need to delete some rows and columns of B so that the resulting submatrix contains only 1's and that the number of deleted entries is smallest. This latter problem is the biclustering problem. □

We now discuss the complexity of the different variants of the biclustering problem explaining the entries in Table I.

Let the $0-1$ matrix B be considered as an adjacency matrix of a bipartite graph $G = (V_1 \cup V_2, E)$. The goal of finding a submatrix of 1s is then equivalent to finding a complete bipartite subgraph in the bipartite graph G. A complete bipartite subgraph in a bipartite graph is known as a *biclique*. Maximizing the sum of the number of columns and rows in the submatrix is the equivalent to finding a biclique with maximum number of nodes. The complementary problem is to minimize the number of deleted nodes in the bipartite graph so that the induced subgraph over the remaining nodes is a biclique. This problem, which we name Biclustering-nodes, is formulated as follows.

$$\begin{array}{lll} \text{(Biclustering-nodes) min} & \sum_{i=1}^{m} r_i + \sum_{j=1}^{n} c_j & \\ \text{subject to} & r_i + c_j \geq 1 & \forall (i,\, j) \text{ such that } b_{ij} = 0 \\ & r_i, c_j \text{ binary} & \forall i,\, j \end{array}$$

Here r_i is a binary variable indicating the deletion of row i of B, and c_j is a binary variable indicating the deletion of column j in B. This problem is polynomially solvable since it is the (unweighted) vertex cover problem in bipartite graph (see also Hochbaum [1998]). It is further solvable in strongly polynomial time by solving a respective minimum s,t-cut problem. Since the weighted vertex cover in a bipartite graph is also

solvable in polynomial time with the same procedure, modifying the objective to be weighted, $\min \sum_{i=1}^{m} w_i r_i + \sum_{j=1}^{n} w'_j c_j$, the biclustering problem remains polynomial-time solvable. Therefore, we can solve directly the maximization problem, of maximizing the sum of the dimensions of the submatrix of all 1's, in polynomial time.

Interestingly, if we want to maximize the number of *elements* in the submatrix, the complexity of the problem changes. It is NP-hard as was shown by Peeters (see Peeters [2003]), and for the *weighted* number of elements, a 2-approximation algorithm was given in Hochbaum [1998]. That means that if the objective changes from the sum of dimensions of the submatrix to the product of the number of rows and columns the problem becomes hard. This is because to maximize the product, the dimensions should be approximately equal, and the submatrix closer to being balanced.

For the objective of minimizing the number of deleted entries valued 1 (or number of deleted edges in the bipartite graph) we present the following formulation. Here $z_{ij} = 1$ if element (i,j) is deleted from the selected submatrix.

$$\begin{array}{lll} \text{(Biclustering-edges) min} & \sum_{i=1}^{m} \sum_{j=1}^{n} z_{ij} & \\ \text{subject to} & r_i + c_j \geq 1 & \forall (i,\, j) \text{ such that } b_{ij} = 0 \\ & z_{ij} \geq r_i & \\ & z_{ij} \geq c_j & \\ & r_i, c_j, z_{ij} \text{ binary} & \forall i,\, j \end{array}$$

This formulation has at most two variables per inequality. Any such problem has a 2-approximation algorithm derived from a minimum s,t-cut problem in a related network, as described in Hochbaum et al. [1993].

In the biclustering problem of finding a largest square submatrix, the graph problem is to find a balanced biclique (that is a biclique with equal sizes on both sides of the bipartition) in a bipartite graph. This problem is known to be NP-hard [Garey and Johnson 1979].

For this problem we suggest the following formulation.

$$\begin{array}{lll} \text{(Balanced biclique) min} & \sum_{j=1}^{n} c_j & \\ \text{subject to} & r_i + c_j \geq 1 & \forall (i,\, j) \text{ such that } a_{ij} = 0 \\ & \sum_{i=1}^{m} r_i \leq m - n + \sum_{j=1}^{n} c_j & \\ & r_i, c_j \text{ binary} & \forall i,\, j \end{array}$$

We leave the analysis of the integrality gap of this formulation as well the question on the existence of a better approximation algorithm for the Biclustering-edges problem for future research.

5. CONCLUSIONS

In this article we consider the Order-Preserving SubMatrix problem. Finding an optimal solution to this optimization problem is important for discovering patterns in gene expression as discussed in earlier works. Our contribution is the design and analysis of approximation algorithms for the problem. Our improved approximation algorithm is a 3-approximation algorithm that runs in $O(m^2 n)$ time. We note that the constants hiding in the O notation are relatively small, and hence our algorithm has reasonable running times for real-life instances.

In this article we used a definition of order-preserving submatrix, that is a $p \times q$ submatrix $B = (b_{ij})$ of A and a permutation of the q columns of B such that all rows have strictly increasing elements. Consider an alternative definition of order-preserving submatrix as a submatrix B with a permutation of its columns such that all rows have

nondecreasing elements. We note that this new definition of order-preserving submatrix does not change the optimization MinOPSM problem in the following sense: Given an instance to the problem with this new definition and a fixed permutation of the columns, we can always add $j\epsilon$ for all entries of column j of the input matrix for a sufficiently small value of ϵ (if all the entries in the input matrix are integer then $\epsilon = \frac{1}{n+1}$ is small enough). In that case, whenever the relaxed constraint $b_{ij_1} \leq b_{ij_2}$ is satisfied we know that $b_{ij_1} < b_{ij_2}$ also holds and vice versa. However, this new definition affects the number of permutations that we need to consider. In this article we use the upper bound m on the number of permutations since the output permutation must coincide with at least one row. If the input matrix to the new definition contains equal numbers, then each row may coincide with multiple permutations, and we need to consider each of these permutations.

REFERENCES

Bar-Yehuda, R. and Even, S. 1981. A linear time approximation algorithm for the weighted vertex cover problem. *J. Algor. 2,* 198–203.

Ben-Dor, A., Chor, B., Karp, R., and Yakhini, Z. 2003. Discovering local structure in gene expression data: The order-preserving submatrix problem. *J. Comput. Biol. 10*, 3–4, 373–384.

Cheng, Y. and Church, G. M. 2000. Biclustering of expression data. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB).* 93–103.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. 2001. *Introduction to Algorithms* 2nd Ed. The MIT Press.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 196.

Hochbaum, D. S. 1982. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput. 11*, 3.

Hochbaum, D. S. 1996. Approximating covering and packing problems: Set cover, vertex cover, independent set and related problems. In *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum Ed., PWS, Boston, 94–143.

Hochbaum, D. S. 1998. Approximating clique and biclique problems. *J. Algor. 29,* 174–200.

Hochbaum, D. S., Megiddo, N., Naor, J. and Tamir, A. 1993. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Math. Program. 62*, 69–83.

Mirkin, B. 1996. *Mathematical Classification and Clustering*. Kluwer.

Peeters, R. 2003. The maximum edge biclique problem is np-complete. *ACM Trans. Discr. Appl. Math.131,* 65-1–65-4.

Received April 2008; revised October 2012; accepted October 2012