Finding Maximum Edge Bicliques in Convex Bipartite Graphs*

Doron Nussbaum[1], Shuye Pu[2], Jörg-Rüdiger Sack[1], Takeaki Uno[3], and Hamid Zarrabi-Zadeh[1]

[1] School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada. Email: {nussbaum,sack,zarrabi}@scs.carleton.ca
[2] Program in Molecular Structure and Function, Hospital for Sick Children, 555 University Avenue, Toronto, Ontario M5G 1X8, Canada. Email: shuyepu@sickkids.ca
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan. Email: uno@nii.ac.jp

Abstract. A bipartite graph $G = (A, B, E)$ is convex on B if there exists an ordering of the vertices of B such that for any vertex $v \in A$, vertices adjacent to v are consecutive in B. A complete bipartite subgraph of a graph G is called a biclique of G. Motivated by an application to analyzing DNA microarray data, we study the problem of finding the maximum edge-cardinality biclique in convex bipartite graphs. Given a bipartite graph $G = (A, B, E)$ which is convex on B, we present a new algorithm that computes the maximum edge-cardinality biclique of G in $O(n \log^3 n \log\log n)$ time and $O(n)$ space, where $n = |A|$. This improves the current $O(n^2)$ time bound available for the problem.

1 Introduction

DNA Microarray technology has recently become a main stream tool in molecular biology for studying the interaction between genes and conditions. Microarray data is often presented as a two-dimensional matrix, where the rows correspond to genes (or clones, open reading frames, etc.), and columns correspond to test conditions (or samples, treatments, time points, etc.). Each entry $[i, j]$ of the matrix represents an expression level of a given gene i measured under a given condition j. Biologists are often capturing the relationships between subsets of genes and subsets of conditions to better understand the biological processes at the cell and the molecular level. For example, biologists are interested in (i) finding a subset of genes that are up-regulated or down-regulated in a coherent fashion under a subset of conditions, or, (ii) finding a subset of conditions (or drugs, diseases) that consistently affect the expression of a subset of genes.

Traditional clustering methods focused on clustering either genes or conditions, but not both, and thus did not provide enough information required by biologists. Biclustering analysis techniques capture the relationship between

* Research supported by NSERC and SUN Microsystems.

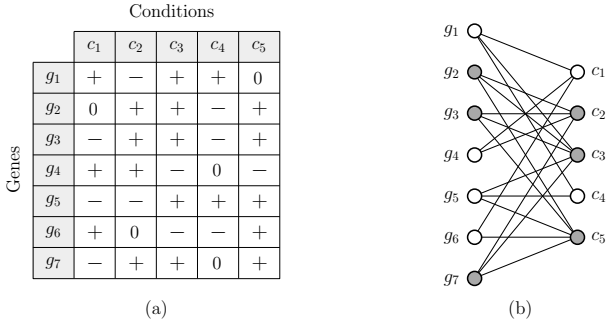

Fig. 1. (a) An exemplary microarray data matrix. Entries are transformed from numerical values to signs to indicate directions of changes of gene expression levels. The symbols + and − represent a significant increase or a significant decrease, respectively, while the symbol $\emptyset$ represents no significant change. In this example the subset of genes $\{g_2, g_3, g_7\}$ and the subset of conditions $\{c_2, c_3, c_5\}$ form a bicluster. (b) A bipartite graph representing significant increases in the microarray data matrix.

genes and conditions and have recently become popular tools in bioinformatics [6]. The objectives of biclustering algorithms is to find a subset of genes $\mathcal{G}$ and a subset of conditions $\mathcal{C}$ such that the change in the expression level of each $g \in \mathcal{G}$ with respect to each $c \in \mathcal{C}$ is significant. The general biclustering problem is NP-hard [22]. Several heuristic biclustering algorithms have been reported in the literature, and most of them are based on greedy and probabilistic approaches [3, 6, 20, 26]. While these algorithms do have technical merits in finding biclusters, they also have various limitations. Thus, more robust biclustering methods are still fervently pursued.

A microarray data matrix is composed of a set of genes $\mathcal{G} = \{g_1, \ldots, g_n\}$ and a set of conditions $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$, such as the one shown in Figure 1(a). This microarray can be modeled as a bipartite graph $G = (A, B, E)$, where each vertex $a_i \in A$ represents a gene $g_i \in \mathcal{G}$ and each vertex $b_j \in B$ represents a condition $c_j \in \mathcal{C}$, and two vertices $a_i \in A$ and $b_j \in B$ are connected, if the expression level of gene g_i significantly increases or significantly decreases under condition c_j (see Figure 1(b)). A bicluster in this graph corresponds to a bipartite subgraph with two vertex sets $S \subseteq A$ and $T \subseteq B$ such that each vertex in S is connected to each vertex in T. Such a complete bipartite subgraph is called a *biclique* of G.

In this paper, we address the problem of finding a maximum bicluster in the expression data, which is equivalent to finding a biclique of maximum edge cardinality in the corresponding bipartite graph. Namely, given a bipartite graph $G = (A, B, E)$, the problem is to find a biclique with two vertex sets $S \subseteq A$ and $T \subseteq B$ such that $|S| \times |T|$ is maximized. This problem is called *maximum edge biclique*, as opposed to the *maximum vertex biclique* problem in which the objective is to maximize $|S| + |T|$. Besides application to molecular biology, maximum

edge biclique has applications to manufacturing optimization [7], formal concept analysis [12], and conjunctive clustering [21].

While maximum vertex biclique is solvable in polynomial time, the maximum edge biclique problem in general bipartite graphs is known to be NP-complete [22]. Indeed, it is hard to approximate the maximum edge biclique in general bipartite graphs to within a factor of n^δ, for some $\delta > 0$ [10, 15] (see also [25] for corresponding inapproximability results in weighted version of the problem). However, for special subclasses of bipartite graphs, such as chordal bipartite graphs and convex graphs, polynomial-time algorithms for the maximum edge biclique problem are available using algorithms that enumerate all maximal bicliques in a given graph [1, 8, 9, 13, 17].

Convex bipartite graphs introduced by Glover [14] naturally arise in several industrial and scheduling applications, and a number of efficient algorithms have been developed on this graph class for problems such as maximum matching, maximum independent set, and minimum feedback vertex set (see e.g. [5, 18, 19, 23, 24]). A bipartite graph $G = (A, B, E)$ is called convex (on B) if there exists an ordering of the vertices of B such that, for any vertex $v \in A$, vertices adjacent to v are consecutive in B. The motivation for studying convex bipartite graphs in the context of biclustering of gene expression data is that a linear ordering of genes exists naturally in several forms, such as chronological ordering in the course of evolution, and spatial ordering on chromosomes.

All the existing algorithms for solving the maximum edge biclique problem are based on enumerating all maximal bicliques in the input graph (see e.g. [1, 9, 17]). It is known that the number of maximal bicliques in a convex bipartite graph with n vertices is $O(n^2)$ [1]. Indeed, it is not hard to construct convex bipartite graphs that have $\Theta(n^2)$ maximal bicliques. Therefore, the existing algorithms for solving the maximum edge biclique problem have a running time of $\Omega(n^2)$ on convex bipartite graphs.

In this paper, we show that the maximum edge biclique problem can be solved more efficiently on convex bipartite graphs by using a pruning technique that avoids enumerating all the maximal bicliques. More precisely, we present a new algorithm that, given a convex bipartite graph $G = (A, B, E)$, computes the maximum edge biclique of G in $O(n \log^3 n \log\log n)$ time and $O(n)$ space, where $n = |A|$. This improves the current $O(n^2)$ time bound available for the problem.

2 Preliminaries

A graph G is *bipartite*, if its set of vertices can be partitioned into two disjoint sets A and B such that every edge of G connects a vertex in A to a vertex in B. We denote such a bipartite graph by $G = (A, B, E)$, where E is the set of edges of G. A complete bipartite subgraph of a graph G is called a *biclique* of G. Given a biclique C of G, we refer to the number of edges in C by the *size* of C, and denote it by $|B|$. Moreover, for a vertex v of G, we denote the set of vertices adjacent to v by $N(v)$.

Let $G = (A, B, E)$ be a bipartite graph. An ordering $\prec$ of B has the *adjacency property* if for every vertex $a \in A$, $N(a)$ consists of vertices that are consecutive (i.e., form an interval) in the ordering $\prec$ of B. A bipartite graph $G = (A, B, E)$ is *convex* if there is an ordering of A or B that fulfills the adjacency property.

For convex graphs, there are linear-time recognition algorithms that output the corresponding orderings on the vertex sets in linear time [2, 4, 16]. Throughout this paper, we assume that the input graph is convex on B, and the vertices of A and B are labeled with integers $1, 2, \ldots$ in the same order imposed by the adjacency property. Figure 2(a) shows an examples of a convex bipartite graph.

In this paper, we denote by $[a\,..\,b]$ the set of integers that lie between two integers a and b, including both. Such a set is called an *integer interval*. Given an integer interval $I = [a\,..\,b]$, the *size* of I, denoted by $|I|$, is the number of integers, $b - a + 1$, contained in I.

3 Problem Transformation

In order to solve the maximum edge biclique problem, we first transform it from a graph theoretical problem to a geometric problem, and then provide an efficient algorithm for solving the geometric problem in Section 4. The main problem considered in this paper is the following:

Problem 1 (Maximum Edge Biclique). Given a convex bipartite graph $G = (A, B, E)$ with $|A| = n$ and $|B| = k$, find a biclique of G that has the maximum number of edges.

We transform the maximum edge biclique problem to a variant of the point dominance problem in the plane. Given two points $p, q \in \mathbb{R}^2$, we say that q is *dominated* by p if $q_x \leqslant p_x$ and $q_y \geqslant p_y$ (in other words, if q lies in a rectangle whose bottom-right corner is at p). Let S be a set of n points in the grid $[1\,..\,k] \times [1\,..\,k]$. We refer to each point of S as a *token*. For a grid point (i, j), we define the *dominance number* of (i, j) w.r.t. S to be $\textsc{dom}(i, j) = |\{(x, y) \in S : (x, y)$ is dominated by $(i, j)\}|$, and define the *magnitude* of (i, j) to be $\textsc{mag}(i, j) = \textsc{dom}(i, j) \times (j - i + 1)$. We call the gird point maximizing $\textsc{mag}(i, j)$ the *maximum point* of the grid.

Problem 2 (Maximum Point). Given a set S of n tokens on a $k \times k$ grid, find a grid point (i, j) that maximizes $\textsc{mag}(i, j)$.

Lemma 1. *Problem 1 is equivalent to Problem 2.*

Proof. By the convexity of G, $N(a)$ is an integer interval in $[1\,..\,k]$ for each vertex $a \in A$. Let π be a function that maps each integer interval $[i\,..\,j] \subseteq [1\,..\,k]$ to a grid point (i, j) on the grid $[1\,..\,k] \times [1\,..\,k]$. For each vertex $a \in A$, we define $\pi(a) \equiv \pi(N(a))$. Let $S = \{\pi(a) : a \in A\}$. We show that finding the maximum edge biclique in G is equivalent to solving the maximum point problem on the set S (see Figure 2).

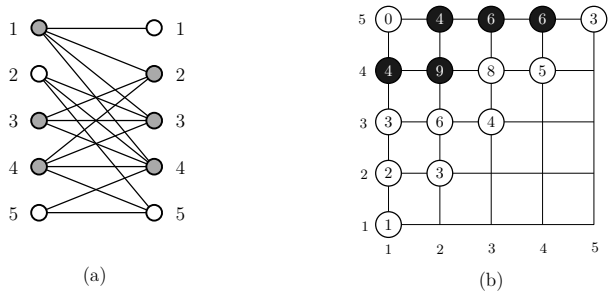

Fig. 2. (a) A convex bipartite graph. (b) The corresponding grid. Tokens are shown in black. The number inside each grid point denotes its magnitude.

The key observation is that for any pair of integer intervals I and R, $R \subseteq I$ if and only if $\pi(I)$ is dominated by $\pi(R)$. This is because $[i\,..\,j] \subseteq [i'\,..\,j']$ if and only if $i' \leqslant i$ and $j' \geqslant j$ which means that (i', j') is dominated by (i, j). Let $R = [i\,..\,j] \subseteq [1\,..\,k]$ represent a set of subsequent vertices in B. Define $A_R = \{a \in A : R \subseteq N(a)\}$. Every vertex in A_R is connected to every vertex in R. Therefore, $A_R \times R$ defines a biclique C_R of G with $|A_R| \times |R|$ edges. Moreover, $|A_R| = |\{a \in A : R \subseteq N(a)\}| = |\{a \in A : \pi(N(a)) \text{ is dominated by } \pi(R)\}| = |\{a \in A : \pi(a) \text{ is dominated by } (i, j)\}| = \textsc{dom}(i, j)$ w.r.t. S. Therefore, $|C_R| = |A_R| \times |R| = \textsc{dom}(i, j) \times (j - i + 1) = \textsc{mag}(i, j)$, and thus, finding a clique of maximum size in G is equivalent to finding a grid point with maximum magnitude. □

Note that $\textsc{mag}(i, j)$ is less than or equal to zero for $j < i$. Therefore, to find the maximum point, we only need to consider grid points (i, j) with $j \geqslant i$.

4 The Algorithm

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of n tokens on a $k \times k$ grid. It is easy to find the maximum point in this grid by a simple scan over the grid points. If the number of tokens is small compared to the grid size, then tokens are sparsely scattered across the grid. An immediate question is whether we can compute the maximum magnitude without necessarily visiting all the grid points. In this section, we answer this question affirmatively by providing an algorithm that finds the maximum point by examining only a small subset (namely, a subquadratic number) of the grid points. We start by the following observation.

Observation 1 *If (x, y) is the maximum point, then there are tokens (x_i, y_i) and (x_j, y_j) in S such that $x_i = x$ and $y_j = y$.*

Proof. Suppose by contradiction that there is no token in S such that $x_i = x$. Let $Q \subseteq S$ be the set of tokens dominated by (x, y), and let $q = (x', y')$ be the token with the largest x-coordinate in Q (ties are broken arbitrarily). Obviously, $\textsc{dom}(x', y) = \textsc{dom}(x, y)$, as there is no token in the rectangle $[x' + 1, x] \times [y, k]$.

Moreover, $x' < x$ by our selection of q. Therefore, $\textsc{dom}(x', y) \times (y - x' + 1) > \textsc{dom}(x, y) \times (y - x + 1)$, which means that $\textsc{mag}(x', y) > \textsc{mag}(x, y)$, contradicting the assumption that (x, y) is the maximum point. Similarly, if there is no token with $y_j = y$, we get into a similar contradiction. □

Observation 1 enables us to restricts the candidates for the maximum point to (x_i, y_j) for some $1 \leqslant i, j \leqslant n$. Let $P = \{x : x = x_i \text{ for some } i\}$ and $Q = \{y : y = y_j \text{ for some } j\}$. Let $s = |P|$ and $t = |Q|$. We denote the elements in P in increasing order by $(p_1, \ldots, p_s)$, and the elements in Q in increasing order by $(q_1, \ldots, q_t)$. The maximum point can be now found via a simple scan over the grid points in $P \times Q$ in $O(n + s \times t) = O(n^2)$ time. (Note that this time bound is independent of the size of the original grid, k.) In the following, we show that this $O(n^2)$ bound can be further improved, using a more clever pruning of the candidate points.

Let $\textsc{d}(i, j) \equiv \textsc{dom}(p_i, q_j)$ be the dominance number, and $\mu(i, j) \equiv \textsc{mag}(p_i, q_j)$ be the magnitude of the grid point (p_i, q_j) in our refined grid. The problem is to find a pair (i, j) for which $\mu(i, j)$ is maximum. We define $\textsc{gap}(i, j, j') = \mu(i, j') - \mu(i, j)$ for $j' \geqslant j$. For example, in the grid shown in Figure 3, $\textsc{gap}(3, 5, 7) = -4$ and $\textsc{gap}(5, 5, 7) = 4$.

Lemma 2. *For indices $i < i'$ and $j < j'$, $\textsc{gap}(i, j, j') \leqslant \textsc{gap}(i', j, j')$ if there is no token at (x, y) such that $p_i < x \leqslant p_{i'}$ and $q_j \leqslant y < q_{j'}$.*

Proof. By the assumption of the lemma we have

$$\textsc{d}(i', j) - \textsc{d}(i, j) = \textsc{d}(i', j') - \textsc{d}(i, j').$$

Observe that

$$\begin{aligned}
\mu(i', j) - \mu(i, j) &= \textsc{d}(i', j) \times (q_j - p_{i'} + 1) - \textsc{d}(i, j) \times (q_j - p_i + 1) \\
&= \textsc{d}(i', j) \times (q_j - p_{i'} + 1) - \textsc{d}(i, j) \times (q_j - p_{i'} + p_{i'} - p_i + 1) \\
&= (\textsc{d}(i', j) - \textsc{d}(i, j)) \times (q_j - p_{i'} + 1) - \textsc{d}(i, j) \times (p_{i'} - p_i) \\
&= (\textsc{d}(i', j') - \textsc{d}(i, j')) \times (q_j - p_{i'} + 1) - \textsc{d}(i, j) \times (p_{i'} - p_i).
\end{aligned}$$

Similarly, we have

$$\mu(i', j') - \mu(i, j') \;=\; (\textsc{d}(i', j') - \textsc{d}(i, j')) \times (q_{j'} - p_{i'} + 1) - \textsc{d}(i, j') \times (p_{i'} - p_i).$$

Therefore,

$$\begin{aligned}
&\textsc{gap}(i', j, j') - \textsc{gap}(i, j, j') \\
&\qquad = (\mu(i', j') - \mu(i', j)) - (\mu(i, j') - \mu(i, j)) \\
&\qquad = (\mu(i', j') - \mu(i, j')) - (\mu(i', j) - \mu(i, j)) \\
&\qquad = [(\textsc{d}(i', j') - \textsc{d}(i, j')) \times (q_{j'} - p_{i'} + 1) - \textsc{d}(i, j') \times (p_{i'} - p_i)] \\
&\qquad\quad -[(\textsc{d}(i', j') - \textsc{d}(i, j')) \times (q_j - p_{i'} + 1) - \textsc{d}(i, j) \times (p_{i'} - p_i)] \\
&\qquad = (\textsc{d}(i', j') - \textsc{d}(i, j')) \times (q_{j'} - q_j) + (\textsc{d}(i, j) - \textsc{d}(i, j')) \times (p_{i'} - p_i) \\
&\qquad \geqslant 0,
\end{aligned}$$

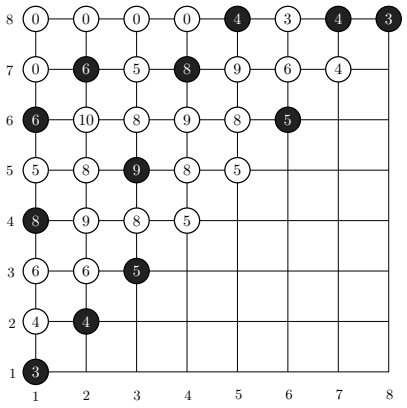

Fig. 3. A refined grid with tokens shown in black. The number on each grid point denotes its magnitude.

where the last inequality holds because $\textsc{d}(i',j') \geqslant \textsc{d}(i,j')$, $\textsc{d}(i,j) \geqslant \textsc{d}(i,j')$, $p_{i'} > p_i$, and $q_{j'} > q_j$. □

For a pair of indices j and j' $(j < j')$, we say that a *flip* occurs between j and j' at index $i > 1$ if either

$$\textsc{gap}(i-1,j,j') \geqslant 0 \quad \text{and} \quad \textsc{gap}(i,j,j') < 0,$$

or

$$\textsc{gap}(i-1,j,j') < 0 \quad \text{and} \quad \textsc{gap}(i,j,j') \geqslant 0.$$

A flip satisfying the former condition is called a *type-1 flip*, and a *type-2 flip* otherwise. For example, in the grid shown in Figure 3, there is a type-2 flip between $j = 3$ and $j' = 5$ at $i = 2$, and there is a type-1 flip between $j = 5$ and $j' = 6$ at index $i = 3$.

Lemma 3. *At any index i, a type-1 flip can occur between j and j' $(j < j')$ only if there is a token at (p_i, q_h) for some $j \leqslant h < j'$.*

Proof. This is a direct corollary of Lemma 2. □

Our idea for finding the maximum point is to construct a binary tree that maintains the maximum magnitude for the points having the same x-coordinate, and update the tree by using flips to find the overall maximum. The algorithm constructs a balanced binary tree B whose leaves are indices $1, \ldots, t$ in increasing order. For an internal node x of B, we denote the set of leaves in the subtree rooted at x by $\textsc{des}(x)$, the left child of x by $l(x)$ (corresponding to smaller indices), and the right child of x by $r(x)$ (corresponding to larger indices). Note that $\textsc{des}(x)$ forms an interval of indices. We denote the largest index among $\textsc{des}(x)$ by $u(x)$, and the smallest index by $b(x)$. For an index i, we define $\mathcal{J}_i^x$ to be the index j maximizing $\mu(i,j)$ among all indices in $\textsc{des}(x)$. Ties are broken by choosing the largest index. (See Figure 4.) The algorithm iteratively increases i one by one, and at each step i, updates $\mathcal{J}_i^x$ when $\mathcal{J}_{i-1}^x \neq \mathcal{J}_i^x$.

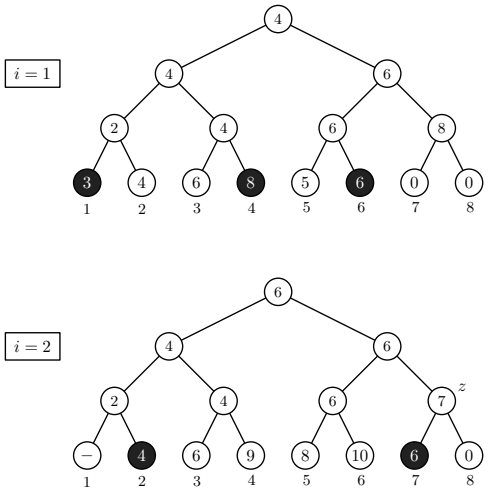

Fig. 4. Binary trees corresponding to the first two columns of the grid depicted in Figure 3. Numbers in the leaves are magnitudes. For each internal node x, the number shown in x is $\mathcal{J}_i^x$.

Observation 2 *$\mathcal{J}_{i-1}^x \neq \mathcal{J}_i^x$ only if either* (a) *$\mathcal{J}_{i-1}^{l(x)} \neq \mathcal{J}_i^{l(x)}$*, (b) *$\mathcal{J}_{i-1}^{r(x)} \neq \mathcal{J}_i^{r(x)}$*, *or* (c) *there is a flip between $\mathcal{J}_{i-1}^{l(x)}$ and $\mathcal{J}_{i-1}^{r(x)}$ at index i.*

By this observation, cases (a) and (b) need a descendant of x which involves case (c). Therefore, to find updates for all nodes in B, we only have to find occurrences of case (c). In the example shown in Figure 4, for the node labeled z we have $\mathcal{J}_1^z \neq \mathcal{J}_2^z$ because there is a type-1 flip between leaves 7 and 8 at $i = 2$. Moreover, for the root of the tree, r, we have $\mathcal{J}_1^r \neq \mathcal{J}_2^r$ because a type-2 flip occurs between leaves 4 and 6 at $i = 2$.

In each step i, we say that a type-1 *event* (resp., a type-2 event) occurs at a node x, if there is type-1 (resp., type-2) flip between $\mathcal{J}_i^{l(x)}$ and $\mathcal{J}_i^{r(x)}$ at index i. We also say that a type-0 event occurs at a node x in step i if there is a token at (p_i, q_j) for some $\mathcal{J}_i^{l(x)} \leqslant j < \mathcal{J}_i^{r(x)}$. By Lemma 3, any type-1 event is also a type-0 event. Therefore, we only need to consider type-0 and type-2 events.

To find events in each step efficiently, we keep the earliest coming event for each internal node. More precisely, for each node x and index i, we define the *next event* of x by the smallest index i' such that $i' > i$ and either a type-0 or a type-2 event occurs at x in step i'. If no such index i' exists, we say that x has no next flip. In each i-th iteration, we maintain the next event for each internal node, and thus, we can find all nodes at which an event occurs in each step by looking only at the next events.

The complete procedure for computing the maximum point is presented in Algorithm 1. In this algorithm, a node x called an *update node* if an event occurs at some descendant of x.

Algorithm 1 FINDMAXPOINT(S)

Initialize:
  build a binary tree B with leaves $1, \ldots, t$
  set $\mathcal{J}_1^\ell = \ell$ for each leaf ℓ of B
  mark all internal nodes of B as update nodes for step $i = 1$
for $i = 1$ to s **do**
  for each update node x at step i in the bottom-up order **do**
    compute $\mu(i, \mathcal{J}_i^{l(x)})$ and $\mu(i, \mathcal{J}_i^{r(x)})$
    update $\mathcal{J}_i^x$
    compute the next event of x
return $\max_{i=1}^{s} \mu(i, \mathcal{J}_i^{root(B)})$

Lemma 4. *The total number of update nodes during the execution of Algorithm 1 is $O(n \log^2 n)$.*

Proof. The number of ancestors of a leaf is $O(\log n)$, thus the number of type-0 events is $O(n \log n)$ in total during the execution of the algorithm. We show below that the same bound applies to the number of type-2 events.

Consider the sequence of events occurring at a node x. We show that in this sequence, there is at most one type-2 event between any two consecutive type-0 events. Fix a node x, and consider two consecutive type-0 events occurring at x, say in steps a and b. Since no other type-0 event occurs in between a and b, by Lemma 3 there is no token at (p_i, q_j) for all $a < i < b$ and $b(x) \leqslant j < u(x)$.

Let c be the first step, $a < c < b$, at which a type-2 event occurs at x. We prove that for all subsequent steps i, $c < i < b$, no type-2 event can occur at x. The proof is based on the following claim:

Claim. For all $c < i < b$, $\mu(i, \mathcal{J}_i^{l(x)}) \leqslant \mu(i, \mathcal{J}_i^{r(x)})$.

To show this, fix an i such that $c < i < b$. We have

$$\mu(c, \mathcal{J}_i^{l(x)}) \;\leqslant\; \mu(c, \mathcal{J}_c^{l(x)}) \;\leqslant\; \mu(c, \mathcal{J}_c^{r(x)}), \tag{1}$$

where the right-hand inequality holds because a type-2 flip has occurred at x in step c, and the left-hand inequality holds because $\mathcal{J}_c^{l(x)}$ points to the maximum leaf in DES($l(x)$) in step c. Using (1) and Lemma 2 we get

$$\mu(i, \mathcal{J}_i^{l(x)}) \;\leqslant\; \mu(i, \mathcal{J}_c^{r(x)}), \tag{2}$$

because there is no token at (p_h, q_j) for all $c < h \leqslant i$ and $b(x) \leqslant j < u(x)$. Using (2) and the fact that $\mu(i, \mathcal{J}_c^{r(x)}) \leqslant \mu(i, \mathcal{J}_i^{r(x)})$, we obtain the claim statement.

It thus follows that the number of type-2 events does not exceed the number of type-0 events, and therefore, we have $O(n \log n)$ events in total. Since each event can be involved in at most $O(\log n)$ update nodes, the total number of update nodes during the execution of the algorithm is $O(n \log^2 n)$. □

Theorem 1. *Algorithm 1 solves the maximum point problem for a set of n tokens on a grid in $O(n \log^3 n \log\log n)$ time and $O(n)$ space.*

Proof. The correctness of the algorithm follows directly from the fact that at each step i, $\mathcal{J}_i^{root(B)}$ maintains the location of the maximum point in column i, and therefore, line 7 of the algorithm returns the maximum point in the whole grid. The running time of Algorithm 1 is dominated by the time needed by the two for-loops. The computation of the magnitude $\mu(i,j)$ in line 4 involves computing $\textsc{d}(i,j)$ which can be done in $O(\log\log n)$ time using an orthogonal range search on the refined grid [11]. For any node x, the computation of the next type-0 event involves finding the earliest coming token in the rectangle $[i\,..\,s]\times[\mathcal{J}_i^{l(x)}\,..\,\mathcal{J}_i^{r(x)}]$, and the computation of the next type-2 event involves a binary search on index i with the computation of $O(\log n)$ magnitudes, both can be done in $O(\log n \log\log n)$ time. By Lemma 4, the two for-loops together iterate $O(n\log^2 n)$ times. Therefore, the running time of the algorithm is $O(n\log^3 n\log\log n)$ in total. The binary tree has $O(n)$ nodes and each node requires memory of constant size to keep the maximum index and the next event. The space complexity is therefore $O(n)$. Note that the initialization step involves sorting tokens by their integer coordinates, constructing the binary tree B, and initializing the data structure for orthogonal range search, all of which can be done in $O(n)$ time and $O(n)$ space. The proof of the theorem is thus complete. □

The following is a direct corollary of Lemma 1 and Theorem 1.

Theorem 2. *Given a convex bipartite graph $G=(A,B,E)$, the maximum edge biclique of G can be computed in $O(n\log^3 n\log\log n)$ time and $O(n)$ space, where $n=|A|$.*

5 Discussion and Conclusions

In this paper, we presented an efficient algorithm for solving the maximum edge biclique problem in convex bipartite graphs in $O(n\log^3 n\log\log n)$ time. The objective function used in our algorithm was $\textsc{dom}(i,j)\times(j-i+1)$. The algorithm works as long as the monotonicity of the GAP function is preserved. Therefore, we can generalize the objective function to any arbitrary function of the form $f(\textsc{dom}(i,j))\times g(j-i)$ such that f is monotone increasing in $\textsc{dom}(i,j)$, and g is monotone increasing in j.

Better running times can be obtained for special subclasses of convex bipartite graphs. In particular, we have shown that by reducing the maximum edge biclique problem to the problem of finding the largest-area rectangle inside a simple polygon with certain properties, we can solve the maximum edge biclique problem in biconvex graphs and bipartite permutation graphs in $O(n\alpha(n))$ and $O(n)$ time, respectively, where $n=\min(|A|,|B|)$, and $\alpha(n)$ is the slowly growing inverse of the Ackermann function. Details will appear in the full version.

Some problems remain open. An immediate problem is whether we can improve the running time of the algorithm presented in this paper by removing

(some of) the logarithmic factors. Finding a better algorithm for the maximum edge biclique problem in chordal bipartite graphs (which is a direct supersetclass of convex bipartite graphs) is another interesting open problem.

References

1. G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11–21, 2004.
2. J. ao Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, 88(1-3):325–354, 1998.
3. 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.
4. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.
5. G. Brodal, L. Georgiadis, K. Hansen, and I. Katriel. Dynamic matchings in convex bipartite graphs. In *Mathematical Foundations of Computer Science 2007*, pages 406–417, 2007.
6. Y. Chen and G. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 93–103, 2000.
7. M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001.
8. V. M. Dias, C. M. de Figueiredo, and J. L. Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1-3):240–248, 2005.
9. V. M. Dias, C. M. de Figueiredo, and J. L. Szwarcfiter. On the generation of bicliques of a graph. *Discrete Applied Mathematics*, 155(14):1826–1832, 2007.
10. U. Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 534–543. ACM, 2002.
11. O. Fries, K. Mehlhorn, S. Nher, and A. Tsakalidis. A $\log \log n$ data structure for three-sided range queries. *Inf. Process. Lett.*, 25(4):269–273, 1987.
12. B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, Berlin, 1996.
13. A. Gély, L. Nourine, and B. Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447–1459, 2009.
14. F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistic Quarterly*, 14:313–316, 1967.
15. A. Goerdt and A. Lanka. An approximation hardness result for bipartite clique. In *Technical Report 48, Electronic Colloquium on Computation Complexity*. 2004.
16. M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000.
17. T. Kloks and D. Kratsch. Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph. *Information Processing Letters*, 55(1):11–16, 1995.

18. Y. D. Liang and M. Chang. Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs. *Acta Informatica*, 34(5):337–346, 1997.
19. W. Lipski and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15(4):329–346, 1981.
20. S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE ACM Transactions on Computational Biology and Bioinformtics*, 1(1):24–45, 2004.
21. N. Mishra, D. Ron, and R. Swaminathan. On finding large conjunctive clusters. In *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory*, pages 448–462, 2003.
22. R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.
23. J. Soares and M. Stefanes. Algorithms for maximum independent set in convex bipartite graphs. *Algorithmica*, 53(1):35–49, 2009.
24. G. Steiner and J. S. Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers & Mathematics with Applications*, 31(12):91–96, 1996.
25. J. Tan. Inapproximability of maximum weighted edge biclique and its applications. In *Theory and Applications of Models of Computation*, pages 282–293. 2008.
26. A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(Supplement 1):S136–S144, 2002.