Finding Maximum Edge Bicliques in Tree Convex Graphs

Vahidreza Afreshteh and Hamid Zarrabi-Zadeh

Department of Computer Science and Engineering
Sharif University of Technology
Tehran, Iran

Abstract

A bipartite graph is said to be *tree convex* if there exists a tree T defined over one part such that the neighborhood of every vertex in the other part forms a connected subtree of T. We study the problem of finding a *maximum edge biclique*—namely, a complete bipartite subgraph containing the largest possible number of edges—in tree convex bipartite graphs. For the special case of *triad convex* graphs, where T has exactly three leaves, we present an $O(n^2 \operatorname{polylog} n)$-time algorithm, significantly improving the previous best $O(n^5)$-time solution. More generally, for tree convex graphs where the underlying tree has a constant number d of leaves, we present an $O(n^{d-1} \operatorname{polylog} n)$-time algorithm, which improves the best previously known runtime by a factor of $n^3/\operatorname{polylog} n$ for any constant $d \geqslant 3$.

1 Introduction

A bipartite graph $G = (A \cup B, E)$ is said to be *tree convex* if there exists a tree T whose vertex set is B, such that for every vertex $v \in A$, the set of its neighbors in B forms a connected subtree of T. See Figure 1 for an illustration. Given a bipartite graph G, the *maximum edge biclique problem* asks for a complete bipartite subgraph—referred to as a *biclique*—that contains the largest possible number of edges in G. The maximum edge biclique problem has attracted considerable attention, primarily due to its applications in biclustering data analysis [5, 10, 34]. In this paper, we focus on solving this problem in the setting of *tree convex bipartite graphs*, where one side of the bipartition exhibits a structural constraint defined by a tree. The problem is formally defined as follows.

Problem 1 (Maximum Edge Biclique in Tree Convex Graphs)**.** *Given a tree T on a vertex set B and a bipartite graph $G = (A \cup B, E)$ that is convex on T, find a biclique in G with the maximum number of edges.*

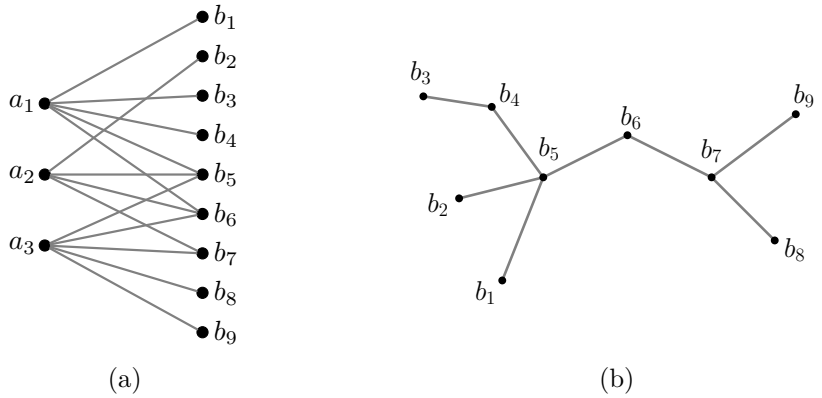


Figure 1: (a) A tree convex bipartite graph. (b) The underlying tree T. The neighbors of vertex a_1, i.e., $\{b_1, b_3, b_4, b_5, b_6\}$ form a connected subtree of T.

Chen and Liu [12] studied the problem of finding maximum edge bicliques in tree convex graphs, and proved that the problem is NP-hard even for *star convex graphs*, where the underlying tree is a star graph. While the general case remains NP-hard, they presented an $O(n^{d+2})$-time algorithm for the case where the underlying tree has a constant number of d leaves.

In this paper, we revisit the maximum edge biclique problem in tree convex bipartite graphs and develop significantly faster algorithms that improve upon the previously best-known results. In particular, for *triad convex* graphs, where the underlying tree T has exactly three leaves, we present an $O(n^2 \operatorname{polylog} n)$-time algorithm, substantially improving the previous $O(n^5)$ algorithm by Chen and Liu [12]. More generally, when the underlying tree has d leaves, for any constant integer $d \geqslant 3$, we present an $O(n^{d-1} \operatorname{polylog} n)$-time algorithm, which improves upon the existing $O(n^{d+2})$-time solution by a factor of $n^3 / \operatorname{polylog} n$.

Our approach builds on a novel geometric formulation of the problem. The key idea is to exploit the hierarchical structure of the underlying tree T, decomposing it into a bounded number of disjoint branches, and reducing the biclique search to a dominance maximization problem over structured point sets. Specifically, we encode the input graph as a compact set of points derived from the convexity imposed by T. This encoding maps the space of connected subtrees of T—each corresponding to a potential biclique—into a low-dimensional grid, where the dimension is determined by the number of leaves in T. By leveraging this bounded dimensionality, our method enables efficient evaluation of candidate solutions through multi-dimensional dominance counting. This transformation allows us to reformulate the original graph-theoretic problem as a geometric optimization problem over the grid points. To solve this efficiently, we partition the grid into slices along one dimension and design a specialized data structure to maintain and query the optimal solution within each slice. We then develop an efficient update mechanism to propagate this

structure across adjacent slices, enabling us to explore the entire grid space while preserving optimality at each step.

1.1 Related work

Three fundamental variants of the maximum biclique problem have received considerable attention in the literature: (i) the *maximum node biclique*, (ii) the *maximum balanced biclique*, and (iii) the *maximum edge biclique*.

In the *maximum node biclique* problem, the objective is to find a complete bipartite subgraph with the largest total number of vertices. Garey and Johnson [17] showed that this problem is solvable in polynomial time (see also [2, 13] for alternative proofs). Moreover, they showed that the weighted version also admits a polynomial-time solution. This follows from its equivalence to the maximum weight independent set problem in bipartite graphs, which is known to be solvable in polynomial time [13].

In contrast, the *maximum balanced biclique* problem, which imposes a symmetry constraint by requiring both sides of the biclique to have equal cardinality, is NP-hard [2, 17]. Furthermore, for any constant $\varepsilon > 0$, the problem is hard to approximate within a factor of n^{ε} under various assumptions [15, 16, 22], and even within $n^{1-\varepsilon}$ [28].

The *maximum edge biclique* problem is considerably more challenging. Its NP-hardness was first established by Dawande et al. [14] in the weighted setting, and later extended to the restricted $\{-1, 1\}$ weighted case [36]. Finally, Peeters [33] showed that the problem remains NP-hard even in the unweighted setting. Beyond hardness, the problem is also notoriously difficult to approximate. Feige [15] first showed that no polynomial-time algorithm can approximate the optimum within a factor of n^{ε}, for any $\varepsilon > 0$ (see also [3, 16] for the same result under weaker assumptions). This bound was later strengthened to $n^{1-\varepsilon}$ under a modified version of the Unique Games Conjecture [4]. The same inapproximability result was also established under the Small Set Expansion Hypothesis, and under the assumption that NP $\not\subseteq$ BPP [28]. Notably, it was posed as an open question in [3] whether a similar inapproximability result could be established under the assumption that P = NP and to the best of our knowledge, this question remains unresolved. Furthermore, the $n^{1-\varepsilon}$ inapproximability bound is tight, as trivial algorithms exist that achieve an n-approximation for both the maximum edge biclique and maximum balanced biclique problems [28].

Despite these negative results, efficient algorithms exist for several structured graph classes. A bipartite graph is called a *convex* if there exists an ordering of the vertices in one part such that the neighbors of every vertex in the other part appear consecutively in that ordering. For convex bipartite graph, Nussbaum et al. [30] addressed an $O(c_n n \log^3 n)$-time algorithm, where c_n denotes the sub-logarithmic time required for range counting queries. They further presented faster algorithms for the *biconvex* and *permutation* subclasses, achieving runtimes of $O(n\alpha(n))$ and

$O(n)$, respectively, where $\alpha(n)$ is the inverse Ackermann function.

Motivated by these results, recent work has explored broader structured families, such as *tree convex bipartite graphs*, where each neighborhood forms a connected subtree of a fixed tree. Chen and Liu [12] showed that the complexity in this setting depends on the structure of the underlying tree. In particular, they proved that the problem is NP-hard for *star convex graphs*, where the underlying tree is a star graph, i.e., a tree in which one central vertex is connected to all other vertices. While the general case remains NP-hard, they presented an $O(n^{d+2})$-time algorithm for the case where the underlying tree has d leaves.

A major variety of other problems and their variants have also been extensively studied in the context of tree convex bipartite graphs, including independent set [25], induced matching [31], feedback vertex set [21, 38], Hamiltonicity and treewidth [11], and domination set [11, 23, 24, 26, 27, 32, 35, 39]. Notably, the complexity of these problems often depends on the number of leaves in the underlying tree. For example, the feedback vertex set problem is NP-hard in general tree convex bipartite graphs [38], but polynomial-time algorithms have been developed for the more restricted triad convex case, even for weighted instances [21].

The problem of determining whether a bipartite graph is tree convex can be reduced to finding a support graph for a hypergraph, namely, a graph on the same vertex set in which every hyperedge induces a connected subgraph. Several linear-time algorithms exist for computing path supports [7, 18, 19, 29] and cycle supports [37]. A bounded-degree tree support with a specified maximum degree for each vertex can be constructed in $O(n^3)$ time [9]. Building on this, a (t, Δ)-tree support, i.e., a support tree with at most t vertices of degree at least 3 and maximum degree Δ, can be testified in $O(n^{t+3})$ time [6]. Note that a tree with d constant leaves qualifies as a (t, Δ)-tree if both t and Δ are bounded. Furthermore, the existence of a cactus support, where the support is a tree of edges and cycles, can also be tested in polynomial time [8]. All of these algorithms are constructive and return a support if one exists.

2 Preliminaries

Let $\mathcal{F}$ be a family of graphs. A bipartite graph $G = (A \cup B, E)$ is called *$\mathcal{F}$-convex*, if there exists a graph $H \in \mathcal{F}$ on the vertex set B such that the neighbors of each vertex in A induce a connected subgraph of H. We call H the *underlying graph*. A bipartite graph G is called *tree convex* if it is $\mathcal{F}$-convex for the family $\mathcal{F}$ of trees. Throughout this paper, we assume that the underlying graph (which is a tree T in our case) is given as part of input.

We denote by $[n]$ the set of integers from 1 and n, inclusively. Given two vectors $X = (x_1, \ldots, x_k)$ and $Y = (y_1, \ldots, y_j)$, we denote by (X, Y) the combined vector $(x_1, \ldots, x_k, y_1, \ldots, y_j)$. We use (X, Y, Z) to combine three vectors in a similar way.

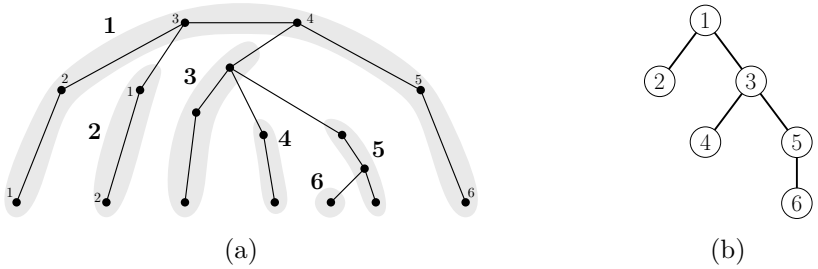


Figure 2: (a) A tree with highlighted decomposed branches, annotated with branch labels and node indices for the first and second branches. (b) The hierarchical structure among the decomposed branches.

3 Problem Transformation

In this section, we present a mapping that encodes subtrees of the underlying tree T as points in a d-dimensional space. This transformation is central to our algorithmic solution, enabling us to reformulate the maximum edge biclique problem as a geometric problem over a structured grid. We begin by describing a tree decomposition procedure, which forms the basis of this mapping.

3.1 Tree decomposition

Let T be a tree with d leaves. We assume that a fixed ordering is defined on the leaves of T. We recursively decompose T into $(d-1)$ node-disjoint paths, which we refer to as *branches*. For each branch, we assign a local *index* to its nodes. The first branch is defined by selecting the two leaves u and v that appear earliest in the specified leaf ordering. Let π denote the path from u to v, inclusive. We assign index 1 to node u, and increment the index sequentially along the path π as we traverse toward node v.

Since T is a tree, removing the path π results in a collection of disjoint subtrees. Let $r \in \pi$ be the first node (along the ordering from u to v) with degree greater than 2. Among the resulting subtrees formerly connected to r, consider one such subtree T' that was attached to r via a node u'. From the set of leaves of T', we select a leaf $v' \neq u'$ that appears earliest in the global leaf ordering, and define a new branch as the path from u' to v'. We assign local indices to the nodes along this branch starting at u' with index 1 and increasing along the path toward v'. This recursive process is then applied to T' and subsequently to each remaining subtree in the collection.

Figure 2 illustrates the result of our decomposition procedure. Each branch is labeled, and node indices are explicitly shown for the first and second branches. As a result of this process, every node $u \in T$ is assigned a unique pair consisting of its branch and index numbers, which we denote by $\text{branch}(u)$ and $\text{index}(u)$, respectively.

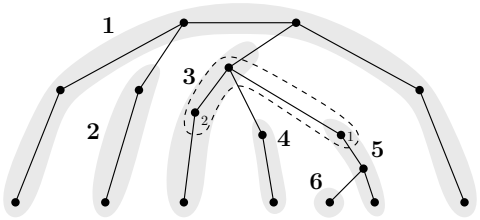


Figure 3: A connected subtree, indicated by dashed lines, mapped to the point $(1, 0, 0, 2, 0, 1, 0)$.

For each branch b, let $\mathrm{desc}(b)$ denote the set of branches that descend from b, either directly or through a chain of intermediate branches. As depicted in Figure 2b, for instance, we have $\mathrm{desc}(3) = \{4, 5, 6\}$. Our construction ensures a hierarchical ordering among branches, meaning that if $b' \in \mathrm{desc}(b)$, then $b < b'$. Furthermore, this ordering is contiguous, i.e., for any branch c such that $b < c < b'$, it follows that $c \in \mathrm{desc}(b)$ as well.

3.2 Mapping subtrees to d-dimensional space

Let S be a connected subtree of T. We define the *primary branch* of S, denoted by $\beta(S)$, as the branch with the smallest number that intersects S. Let B_i denote the set of nodes that lie on the ith branch. We define a mapping μ that associates each connected subtree S of T with a point $(x_0, \ldots, x_{d-1})$ as follows:

$$x_i = \begin{cases} \min\{\mathrm{index}(v) \mid v \in \beta(S) \cap S\} & \text{if } i = 0, \\ \max\{\mathrm{index}(v) \mid v \in B_i \cap S\} & \text{if } i \geqslant 1 \text{ and } B_i \cap S \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, x_0 encodes the first endpoint of S on its primary branch, while each x_i for $i \geqslant 1$ records the extent of S along branch i. It is easy to verify that μ is injective: every connected subtree of T is uniquely determined by its set of leaves, whose branch and local indices are encoded in the vector $(x_0, \ldots, x_{d-1})$. For example, in Figure 3, the point $(1, 0, 0, 2, 0, 1, 0)$ uniquely represents a subtree whose primary branch is 3, and whose leaves appear at index 2 on branch 3, and index 1 on branch 5, respectively.

Let $\mathcal{T}$ denote the set of all connected subtrees of T, and let $\mathcal{P} = \mu(\mathcal{T})$ be the image of these subtrees under the mapping μ. The tree convexity of the bipartite graph $G = (A \cup B, E)$ with respect to tree T implies that for every vertex $v \in A$, the set of neighbors of v in B induces a connected subtree of T. We denote this subtree by $\tau(v)$ for each $v \in A$. Let $\mathcal{Q} \subseteq \mathcal{P}$ be the set of points corresponding to these subtrees, i.e., $\mathcal{Q} = \{\mu(\tau(v)) \mid v \in A\}$. This provides a compact geometric representation of the tree convex structure of G.

Remark 1. *A tree convex graph can be succinctly represented by the point set $\mathcal{Q}$ using only $O(n)$ space, in contrast to the standard adjacency-based representation which requires $O(n^2)$ space.*

3.3 Problem reformulation

The following lemma allows us to restrict our attention to connected subtrees of T when searching for a maximal edge biclique.

Lemma 1. *Let $G = (A \cup B, E)$ be a bipartite graph that is convex over a tree T. Let $C = (A' \cup B', E')$ be a maximal edge biclique in G. Then, the subgraph of T induced by B' is connected.*

Proof. Suppose for contradiction that the subgraph S of T induced by B' is disconnected. Then there exist vertices $u, v \in B'$ that lie in different components of S, and thus no path between them exists within S. However, since T is a tree, there is a unique path π in T connecting u and v.

By the tree convexity of G, for any vertex $w \in A'$ which is adjacent to both u and v, all vertices along the path π are also adjacent to w in G. Moreover, because C is a biclique, every vertex in A' is adjacent to every vertex in B', including u and v. Therefore, all vertices along π are adjacent to all vertices in A', and thus the set $B' \cup V(\pi)$ can be extended to a larger biclique $C' = (A' \cup (B' \cup V(\pi)), E'')$ with more edges than C, contradicting the maximality of C. □

By Lemma 1, any maximal edge biclique corresponds to a connected subtree $S \in \mathcal{T}$. Via the injective mapping μ, we may instead search over the corresponding point set $\mathcal{P} = \mu(\mathcal{T})$.

Recall that for any connected subtree S of the tree T, the primary branch $\beta(S)$ is defined as the branch with the smallest number that intersects S. We extend this notation and write $\beta(X)$ for a point $X = (x_0, \ldots, x_{d-1}) \in \mathcal{P}$, to denote the smallest index $i \geq 1$ such that $x_i > 0$.

Let $C = (A' \cup B', E')$ be a maximal biclique in G. Since the bipartite subgraph C is complete, the number of edges is $|A'| \cdot |B'|$. By Lemma 1, the set B' induces a connected subtree S of T, and let $X = \mu(S)$ be its corresponding point representation. The size of B' (i.e., the number of nodes in S) can be computed as:

$$w(X) = \left(\sum_{i=1}^{d-1} x_i \right) - x_0 + 1. \tag{1}$$

To compute $|A'|$, we count the number of vertices $v \in A$ such that the subtree $\tau(v)$ (i.e., the subtree induced by its neighbors in B) contains S. Let $X' = \mu(\tau(v))$ and $b = \beta(X)$. We identify two cases under which $\tau(v)$ covers S:

(i) $b = \beta(X')$ and $x'_i \geq x_i$ for all $i \geq 1$, and $x'_0 \leq x_0$,

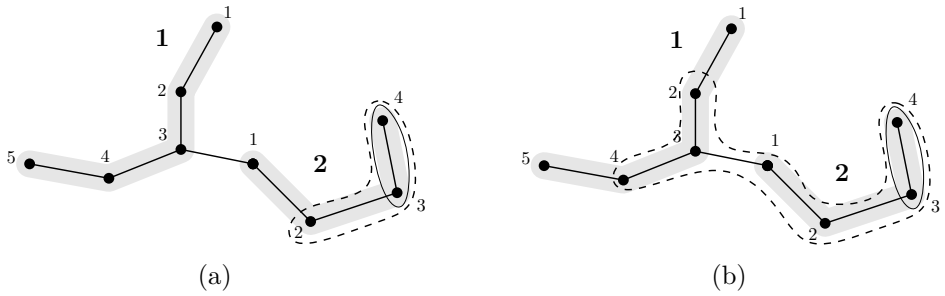


Figure 4: A triad tree with decomposed branches, showing a subtree X' (solid lines) and two subtrees (dashed lines) that cover it: (a) a covering subtree with the same primary branch; (b) a covering subtree from a higher branch in the hierarchy.

(ii) $b \in \text{desc}(\beta(X'))$ and $x'_i \geq x_i$ for all $i \geq 1$.

See Figure 4 for an illustration. These two cases correspond to the following sets:

$$\mathcal{D}(X) = \{X' \in \mathcal{Q} \mid \forall i \in [d-1],\ x'_i \geq x_i,\ \beta(X') = \beta(X),\ x'_0 \leq x_0\}, \tag{2}$$

$$\mathcal{D}'(X) = \{X' \in \mathcal{Q} \mid \forall i \in [d-1],\ x'_i \geq x_i,\ \beta(X) \in \text{desc}(\beta(X'))\}. \tag{3}$$

Now, we can show the number of vertices in A whose neighbor-induced subtrees cover S by the following function:

$$\delta(X) = |\mathcal{D}(X)| + \left|\mathcal{D}'(X)\right|. \tag{4}$$

Note that the function $\delta(X)$ is non-decreasing on x_0, and is non-increasing on all other coordinates. The sets $\mathcal{Q}, \mathcal{D}(X)$, and $\mathcal{D}'(X)$ should be considered as multisets, as multiple vertices in A may map to the same subtree. Combining the formulations from Equations (1) and (4), we define our main objective function, which quantifies the size of a biclique corresponding to a subtree encoded by point X, as follows:

$$f(X) = \delta(X) \cdot w(X). \tag{5}$$

Theorem 2. *Let $G = (A \cup B, E)$ be a bipartite graph that is convex over a tree T, and let $C = (A' \cup B', E')$ be a maximum edge biclique in G. If $X \in \mathcal{P}$ is a point maximizing f, then $|A'| \times |B'| = f(X)$.*

Proof. Let S be the connected subtree of T induced by B', and let $X = \mu(S)$. As shown above, the size of B' is $w(X)$ and the number of vertices $v \in A$ such that $\tau(v)$ covers S is $\delta(X)$. Therefore, the number of edges in the corresponding biclique is $f(X) = \delta(X) \cdot w(X)$. Maximizing this product over all $X \in \mathcal{P}$ yields a maximum edge biclique in G. □

Theorem 2 thus reduces the task of finding a maximum edge biclique in a tree convex bipartite graph to the problem of maximizing the objective function $f(X)$ over the point set $\mathcal{P}$.

Problem 2. *Given a set of points $\mathcal{P}$ in d dimensions, find a point $X \in \mathcal{P}$ that maximizes $f(X) = \delta(X) \cdot w(X)$.*

3.4 Problem relaxation

As mentioned earlier, our goal is to find a point in $\mathcal{P}$ that maximizes the function f. However, directly enumerating all points in $\mathcal{P}$ is nontrivial, since the mapping μ used to define $\mathcal{P}$ depends intricately on the hierarchical branch decomposition of T, which introduces additional structural constraints that must be tracked and preserved during enumeration. To overcome this difficulty, we introduce a structured superset $\mathcal{P}^+ \supseteq \mathcal{P}$, along with a corresponding collection of subgraphs $\mathcal{T}^+ \supseteq \mathcal{T}$, that includes certain additional disconnected subgraphs of T formed by selecting one interval from each branch. By relaxing the original formulation in this way, we enable more efficient exploration of the solution space, while guaranteeing that these additional disconnected subgraphs are never viable candidates for an optimal solution.

Let n_i denote the number of nodes in branch i of T. For each branch $b \in [d-1]$, define the set U_b to consist of points $(x_0, \ldots, x_{d-1})$ such that:

- $x_0 \in [n_b]$,
- $x_j = 0$ for all $1 \leq j < b$, and
- $x_j \in [n_j]$ for all $b \leq j \leq d-1$.

It follows that for any subtree $S \in \mathcal{T}$ with $\beta(S) = b$, the point $\mu(S)$ lies in U_b. Let $I_b = \{(x_0, \ldots, x_{d-1}) \in U_b \mid x_b < x_0\}$ be the set of "invalid" points of U_b. We define $\mathcal{P}^+ = \bigcup_b (U_b \setminus I_b)$.

We now describe how each point $X = (x_0, \ldots, x_{d-1}) \in \mathcal{P}^+$ can be mapped to a (not necessarily connected) subgraph of T. Let $b = \beta(X)$. The corresponding subgraph is constructed by taking:

- the path from node x_0 to x_b along branch b, and
- for each $i \neq b$ such that $x_i > 0$, the path from node 1 to x_i along branch i.

Note that the resulting subgraph may be disconnected; see Figure 5 for an illustration. We extend the definitions of the functions w, δ, and consequently f, to apply to all points in $\mathcal{P}^+$ using the same expressions as before. The following lemma justifies maximizing f over $\mathcal{P}^+$ rather than $\mathcal{P}$ in solving Problem 2.

Lemma 3. *The set $\mathcal{P}$ is contained within $\mathcal{P}^+$, and no point in $\mathcal{P}^+ \setminus \mathcal{P}$ can maximize the objective function f.*

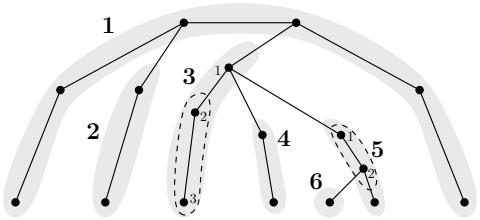


Figure 5: A disconnected subgraph mapped to the point $(2, 0, 0, 3, 0, 2, 0)$.

Proof. Let S be a connected subtree of T, and let $X = \mu(S) = (x_0, \ldots, x_{d-1})$ be its image under the mapping μ. By definition, if the primary branch of S is $b = \beta(S)$, then $X \in U_b$, and we have the constraint $x_0 \leqslant x_b$. This condition ensures that the root index x_0 does not exceed the maximum index on the primary branch b, which must hold for any valid image of a connected subtree. Consequently, X cannot lie in I_b, which is defined to contain points violating this constraint. Therefore, all valid points in $\mathcal{P}$ must lie in $U_b \setminus I_b$ for some b, which implies $\mathcal{P} \subseteq \bigcup_b (U_b \setminus I_b) = \mathcal{P}^+$.

To prove the second statement, suppose, for contradiction, that a point $X' = (x'_0, \ldots, x'_{d-1}) \in \mathcal{P}^+ \setminus \mathcal{P}$ maximizes the function f. Let S' be the disconnected subgraph of T associated with X', and let S be the smallest connected subtree of T that contains all nodes in S'. Let $X = \mu(S) = (x_0, \ldots, x_{d-1})$ be the corresponding valid point in $\mathcal{P}$.

Recall that, by definition of w, $w(X)$ denotes the number of nodes in S. Similarly, according to the definition of $\mathcal{P}^+$, for any point $X' \in \mathcal{P}^+$, $w(X')$ denotes the number of nodes in S'. Since S' is a proper subgraph of S, it follows that $w(X) > w(X')$.

Moreover, by the definitions of $\mathcal{D}$ and $\mathcal{D}'$ in Equations (2) and (3), the following holds. If $\beta(X) = \beta(X')$, then $\mathcal{D}(X) = \mathcal{D}(X')$ and $\mathcal{D}'(X) = \mathcal{D}'(X')$, and thus $\delta(X) = \delta(X')$. Otherwise, $\beta(X') \in \mathrm{desc}(\beta(X))$. In this case, $\mathcal{D}(X')$ is empty, since S' is disconnected, and $\mathcal{D}'(X') = \mathcal{D}'(X) \cup \mathcal{D}(X)$, as S is the smallest subtree that covers S'. Therefore, in both cases, $\delta(X) = \delta(X')$.

Hence, we have $f(X) = \delta(X) \cdot w(X) > \delta(X') \cdot w(X') = f(X')$, which contradicts the assumption that X' maximizes f. Therefore, no point in $\mathcal{P}^+ \setminus \mathcal{P}$ can be optimal, completing the proof. □

By Lemma 3, it suffices to restrict our search to the superset $\mathcal{P}^+$. Moreover, we can partition $\mathcal{P}^+$ into disjoint regions $\mathcal{P}_b^+ = \mathcal{P}^+ \cap U_b$ corresponding to primary branches b, and independently search for the optimal point within each region. This leads to the following subproblem:

Problem 3. *For each branch b, find a point $X \in \mathcal{P}_b^+$ that maximizes the objective function $f(X) = w(X) \cdot \delta(X)$.*

4 The Algorithm

In the previous section, we introduced Problem 3 as a crucial subproblem that paves the way for solving our main objective: finding the maximum edge biclique in tree convex bipartite graphs. In this section, we present an efficient and scalable algorithmic framework for solving Problem 3, which lies at the core of our overall approach.

For the special case of $d = 2$, Problem 3 reduces to a two-dimensional grid optimization problem, which can be solved in $O(n \operatorname{polylog} n)$ time using a binary tree-based search strategy [30]. To address higher-dimensional instances, we generalize this approach by recursively extending the binary tree structure into a multi-level auxiliary tree. This hierarchical data structure efficiently supports updates, allowing us retain the performance of the two-dimensional algorithm while scaling gracefully to any constant dimension. Consequently, our framework achieves a running time of $O(n^{d-1} \operatorname{polylog} n)$ for constant d, providing a practical and theoretically sound solution to high-dimensional instances of the problem.

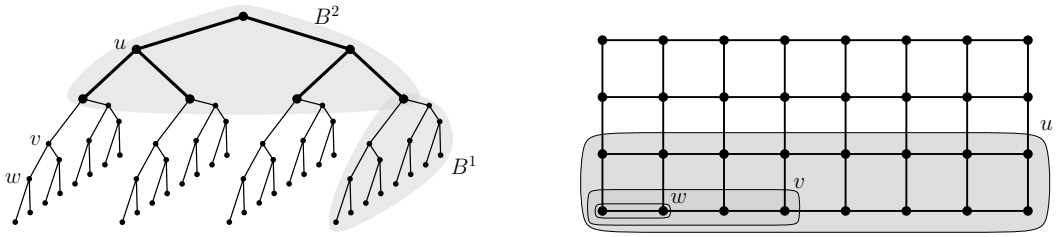
4.1 The auxiliary binary trees

To support efficient search in the multi-dimensional domain $\mathcal{P}^+$, we recursively define a family of auxiliary trees, denoted by B^j, for $j \in [d-1]$. Each B^j is responsible for a j-dimensional subset of the space.

At the base level, B^1 is a balanced binary tree built over the discrete values of the x_0 coordinate (i.e., the first dimension), ordered left to right. For each level $j \geq 2$, the tree B^j is defined recursively as a binary tree whose leaves are the roots of B^{j-1} trees (see Figure 6 for an illustration). These subtrees correspond to discrete values of the coordinate x_{j-1}, and are ordered left to right by increasing x_{j-1}. Thus, the entire structure represents a $(d-1)$-dimensional grid through hierarchical composition.

Each point in the set $U = \bigcup_b U_b$ can be expressed in the form (x, Y, z), where x and z are scalars, and Y is a $(d-2)$-dimensional vector. For a fixed value of z, we construct a separate tree B_z over all points (x, Y) such that $(x, Y, z) \in U_b$, where b is the primary branch for which we are solving Problem 3. Recall that B_z is defined over the set of d-dimensional points, and therefore has the same structure as the auxiliary tree B^{d-1}. Each leaf in B_z is initialized with its corresponding point (x, Y), if the point (x, Y, z) is valid (i.e., not in the invalid set I_b), or marked as inactive otherwise. Each internal node in B_z stores a point among its descendants for which f is maximum. As a result, the root of B_z stores the point maximizing f over all entries with fixed z in the domain U_b.

To solve Problem 3 for branch b, we evaluate the roots of B_z over all z values, returning the point with the highest f-value overall. Note that the values stored in leaves of B_z remain identical across different values of z, while the values of internal nodes may change due to varying $f(x, Y, z)$. Note that we use the notation $f(x, Y, z)$



(a) (b)

Figure 6: (a) An auxiliary binary tree B^2. (b) The corresponding 2-dimensional grid, where each point maps to a leaf of B^2. Each internal node of the tree represents a subregion of the grid; three such regions are highlighted and labeled as u, v, and w.

as a shorthand for $f((x, Y, z))$ for brevity.

4.2 Updating the binary tree

Since the structural layout of each B_z remains consistent across all values of z, we maintain a single binary tree B that is initially configured for $z = 0$ and update it incrementally throughout the algorithm. This allows us to simulate the structure of B_z for any z without rebuilding the tree from scratch. The algorithm progresses through increasing values of z, and at each step z, we efficiently adjust the internal nodes of B to reflect the corresponding tree B_z and determine the maximizing point stored at its root.

Let u be a specific node in B. Denote by u_z and $u_{z'}$ the representations of u at steps z and z', i.e., in B_z and $B_{z'}$, respectively. Since the leaves are invariant across all B_z, changes only affect internal nodes. For an internal node u to differ between B_z and $B_{z'}$, one of the following conditions must hold:

C1. The values stored in the left or right children of u_z and $u_{z'}$ differ.

C2. The children are the same, but the result of the comparison changes. More precisely, if the left child is (x, Y) and the right child is (x', Y'), then one of the following two conditions holds:

C2.1. $f(x', Y', z) \geqslant f(x, Y, z)$ and $f(x', Y', z') < f(x, Y, z')$.

C2.2. $f(x', Y', z) < f(x, Y, z)$ and $f(x', Y', z') \geqslant f(x, Y, z')$.

To simplify our reasoning, we define the *gap* between two points (x, Y, z) and (x', Y', z) as:

$$\Delta_z\big((x, Y), (x', Y')\big) = f(x, Y, z) - f(x', Y', z).$$

Conditions (C2.1) and (C2.2) correspond to a change in the sign of the gap function between z and z', indicating that the comparison result has flipped. Let $Y =$

$(y_1, \ldots, y_j)$ and $Y' = (y'_1, \ldots, y'_j)$ be two j-dimensional vectors. We use $Y \prec Y'$ to denote $y_i < y'_i$ for all $i \in [j]$, and $Y \preceq Y'$ to denote $y_i \leqslant y'_i$ for all $i \in [j]$. To update B from B_z to $B_{z'}$, we perform the following types of updates:

U1. For every point $(x, S, z) \in \mathcal{Q}$, update the ancestors of the leaf corresponding to each (x, Y) such that $Y \preceq S$.

U2. Update any internal node u whose children (x, Y) and (x', Y') induce a sign change in the gap function between z and z' (i.e., satisfy (C2.1) or (C2.2)), and u is not covered by any (U1) update.

U3. Update the ancestors of all nodes identified for (U2) updates, as changes at a child node may propagate upward and affect comparisons at higher levels.

Let u be an internal node of B whose left and right children store (x, Y) and (x', Y'), respectively. If u lies in the bottom layer of B, namely B^1, then by the structure of B, we have $Y = Y'$ and $x < x'$. Suppose that u is not covered by any (U1) update at step z. Then there can be no point $(r, S, z) \in \mathcal{Q}$ such that $x < r \leqslant x'$ and $Y \preceq S$. Indeed, if such a point existed, the leaf corresponding to (r, Y)—which lies in the subtree rooted at u by construction of B—would trigger a (U1) update, contradicting the assumption that u is uncovered. Now suppose u appears in a higher layer of B, i.e., B^i for some $i > 1$, and that u is not covered by any (U1) update at step z. Then there can be no point $(r, S, z) \in \mathcal{Q}$ such that either $Y \preceq S$ or $Y' \preceq S$, for any value of r. If such a point existed, it would trigger a (U1) update via the leaves (r, Y) or (r, Y'), both of which lie in the subtree rooted at u. We summarize these observations as follows.

Property 1. *Let u be an internal node of B^1, whose left and right children store (x, Y) and (x', Y), respectively. If u is not assigned any (U1) update at step z, then there is no point $(r, S, z) \in \mathcal{Q}$ such that $x < r \leqslant x'$ and $Y \preceq S$.*

Property 2. *Let u be an internal node of B^i for some $i > 1$, whose left and right children store (x, Y) and (x', Y'), respectively. If u is not assigned any (U1) update at step z, then there is no point $(r, S, z) \in \mathcal{Q}$ such that either $Y \preceq S$ or $Y' \preceq S$, for any value of r.*

The following lemma, along with its corollary, reveals a key structural property that guarantees our defined updates are sufficient to update B efficiently, without incurring redundant computations.

Lemma 4. *Let $x < x'$ be two integers, Y a fixed $(d-2)$-dimensional vector, and $z < z'$ two integers. If there exists no point $(r, S, t) \in \mathcal{Q}$ such that $x < r \leqslant x'$, $Y \preceq S$, and $z \leqslant t < z'$, then $\Delta_z((x, Y), (x', Y)) \geqslant \Delta_{z'}((x, Y), (x', Y))$.*

Proof. By the definition of functions Δ and f, we have:

$$\begin{aligned}
&\Delta_z\left((x,Y),(x',Y)\right) - \Delta_{z'}\left((x,Y),(x',Y)\right)\\
&\quad = (f(x,Y,z) - f(x',Y,z)) - (f(x,Y,z') - f(x',Y,z'))\\
&\quad = (w(x,Y,z)\,\delta(x,Y,z) - w(x',Y,z)\,\delta(x',Y,z))\\
&\qquad - (w(x,Y,z')\,\delta(x,Y,z') - w(x',Y,z')\,\delta(x',Y,z')).
\end{aligned}$$

Moreover, by the definition of the weight function w, we have $w(x,Y,z') = w(x,Y,z) + (z'-z)$. Similarly, $w(x',Y,z') = w(x',Y,z) + (z'-z)$. Therefore, we obtain:

$$\begin{aligned}
&\Delta_z\left((x,Y),(x',Y)\right) - \Delta_{z'}\left((x,Y),(x',Y)\right)\\
&\quad = w(x,Y,z)\,\delta(x,Y,z) - w(x',Y,z)\,\delta(x',Y,z)\\
&\qquad - \left[(w(x,Y,z) + (z'-z))\,\delta(x,Y,z') - (w(x',Y,z) + (z'-z))\,\delta(x',Y,z')\right]\\
&\quad = w(x,Y,z)\,(\delta(x,Y,z) - \delta(x,Y,z'))\\
&\qquad - w(x',Y,z)\,(\delta(x',Y,z) - \delta(x',Y,z')) + (z'-z)\,(\delta(x',Y,z') - \delta(x,Y,z')).\\
&\quad = (w(x,Y,z) - w(x',Y,z))\,(\delta(x,Y,z) - \delta(x,Y,z'))\\
&\qquad + (z'-z)\,(\delta(x',Y,z') - \delta(x,Y,z')),
\end{aligned}$$

where the last equality holds because $\delta(x,Y,z) - \delta(x,Y,z') = \delta(x',Y,z) - \delta(x',Y,z')$ by the assumption of the lemma. Moreover, since function $\delta(X)$ is non-increasing on z by its definition, we have $\delta(x,Y,z) \geqslant \delta(x,Y,z')$. Similarly, the non-decreasing property of $\delta(X)$ on x implies $\delta(x',Y,z') \geqslant \delta(x,Y,z')$. Additionally, by the definition of the function w, we know that $w(x,Y,z) - w(x',Y,z) = (x'-x)$. Recall that both $(x'-x)$ and $(z'-z)$ are positive. Therefore, we conclude that $\Delta_z\left((x,Y),(x',Y)\right) - \Delta_{z'}\left((x,Y),(x',Y)\right) \geqslant 0$. □

Corollary 5. *Let x, x', z, and z' be integers such that $x < x'$ and $z' = z+1$, and let Y be a $(d-2)$-dimensional vector. If $\Delta_z\left((x,Y),(x',Y)\right) < \Delta_{z'}\left((x,Y),(x',Y)\right)$, then there exists a point $(r,S,z) \in \mathcal{Q}$ such that $x < r \leqslant x'$ and $Y \preceq S$.*

The following lemma shows that our update rules cover all nodes requiring update, i.e., those satisfying conditions (C1), (C2.1), or (C2.2).

Lemma 6. *Let z and z' be two integers with $z' = z+1$. Let u_z and $u_{z'}$ denote internal nodes at the same position in B_z and $B_{z'}$, respectively, and suppose the value of u_z differs from that of $u_{z'}$. Then u_z is covered by at least one of the update types (U1), (U2), or (U3).*

Proof. Consider the tree B at step z, and let u be the node in B corresponding to u_z. Suppose the left and right children of u store (x,Y) and (x',Y'), respectively. We first consider the case where u belongs to the bottom layer of B, namely B^1. In this layer, the structure of B ensures that $Y = Y'$ and $x < x'$. If u satisfies condition (C2.1), then by Corollary 5, there exists a point $(r,S,z) \in \mathcal{Q}$ such that

$x < r \leq x'$ and $Y \preceq S$. Therefore, u is covered by a (U1) update, as guaranteed by Property 1. On the other hand, if u satisfies condition (C2.2) and is not covered by any (U1) update, then it is included in a (U2) update by definition due to the sign flip of the gap function.

Next, suppose that u appears in a higher layer B^i for some $i > 1$. Assume that u satisfies either condition (C2.1) or (C2.2). If there exists a point $(r, S, z) \in \mathcal{Q}$ such that $Y \preceq S$ or $Y' \preceq S$, for any value of r, then u is covered by a (U1) update, as established by Property 2. Otherwise, the sign flip at u fulfills the criteria of (U2) updates.

Finally, if u satisfies condition (C1), then due to the invariance of leaves across all B_z, some internal node in the subtree rooted at u must satisfy either (C2.1) or (C2.2). Therefore, u is covered either by a (U1) update or by a (U3) update. □

Note that updates of type (U1) can be derived directly from the points in $\mathcal{Q}$, and updates of type (U3) follow from updates of type (U2). In the following, we show that the number of (U2) updates is bounded by the number of (U1) updates, and offer a constructive method for identifying all such (U2) updates. To this end, we distinguish two cases based on the level of the internal node being updated. For nodes located in the bottom layer of B, the following corollary derived from Lemma 4 provides the desired bound.

Corollary 7. *Let x, x', z, and z' be integers with $x < x'$ and $z < z'$, and let Y be a fixed $(d-2)$-dimensional vector. Suppose there is no point $(r, Y, t) \in \mathcal{Q}$ such that $x < r \leqslant x'$ and $z \leqslant t < z'$. Then, for every integer i with $z \leqslant i < z'$, the gap function $\Delta_i((x, Y), (x', Y))$ varies monotonically as i increases from z to z'. Consequently, the sign of the gap function can change at most once, when i increases from z to z'.*

This corollary guarantees that each internal node in the bottom layer of B is subject to at most one (U2) update between any two consecutive (U1) updates. A similar bound for internal nodes in higher layers of B will be established in the following lemmas.

Lemma 8. *Let $x < x'$ be integers, Y and Y' be $(d-2)$-dimensional vectors, and $z < z'$ be integers. Assume there is no point $(r, S, t) \in \mathcal{Q}$ with $z \leqslant t < z'$ such that either $Y \preceq S$ or $Y' \preceq S$, for any value of r. Then, for every integer i with $z \leqslant i < z'$, the gap $\Delta_i((x, Y), (x', Y'))$ changes monotonically as i increases from z to z'. Consequently, the sign of the gap function can change at most once, when i increases from z to z'.*

Proof. By the definition of the function δ from Equation (4), we have:

$$\begin{aligned}
&\delta(x, Y, i) \\
&\quad = \left|\{(r, S, t) \in \mathcal{Q} \mid \beta(r, S, t) = \beta(x, Y, i), r \leqslant x, Y \preceq S, i \leqslant t\}\right| \\
&\qquad + \left|\{(r, S, t) \in \mathcal{Q} \mid \beta(x, Y, i) \in \mathrm{desc}(\beta(r, S, t)), Y \preceq S, i \leqslant t\}\right| \\
&\quad = \left|\left\{(r, S, t) \in \mathcal{Q} \mid \beta(r, S, t) = \beta(x, Y, i), r \leqslant x, Y \preceq S, z' \leqslant t\right\}\right| && (6) \\
&\qquad + \left|\left\{(r, S, t) \in \mathcal{Q} \mid \beta(r, S, t) = \beta(x, Y, i), r \leqslant x, Y \preceq S, i \leqslant t < z'\right\}\right| && (7) \\
&\qquad + \left|\left\{(r, S, t) \in \mathcal{Q} \mid \beta(x, Y, i) \in \mathrm{desc}(\beta(r, S, t)), Y \preceq S, z' \leqslant t\right\}\right| && (8) \\
&\qquad + \left|\left\{(r, S, t) \in \mathcal{Q} \mid \beta(x, Y, i) \in \mathrm{desc}(\beta(r, S, t)), Y \preceq S, i \leqslant t < z'\right\}\right|. && (9)
\end{aligned}$$

Since $\beta(x, Y, i) = \beta(x, Y, z')$ for all integers i, by the definition of the function β, the sum of the terms in lines (6) and (8) is precisely $\delta(x, Y, z')$. On the other hand, the two sets in lines (7) and (9) are empty by assumption, because no point in $\mathcal{Q}$ satisfies $Y \preceq S$ and $i \leqslant t < z'$. Hence, their contributions are zero. It follows that $\delta(x, Y, i) = \delta(x, Y, z')$, for all $z \leqslant i < z'$. As such, for any integers i and i' such that $z \leq i < i' < z'$, we have $\delta(x, Y, i) = \delta(x, Y, i') = \delta(x, Y, z')$ and $\delta(x', Y', i) = \delta(x', Y', i') = \delta(x', Y', z')$. Therefore,

$$\begin{aligned}
&\Delta_{i'}\left((x, Y), (x', Y')\right) - \Delta_i\left((x, Y), (x', Y')\right) \\
&\quad = w(x, Y, i')\,\delta(x, Y, z') - w(x', Y', i')\,\delta(x', Y', z') \\
&\qquad - \left[w(x, Y, i)\,\delta(x, Y, z') - w(x', Y', i)\,\delta(x', Y', z')\right] \\
&\quad = \left(w(x, Y, i') - w(x, Y, i)\right)\delta(x, Y, z') \\
&\qquad - \left(w(x', Y', i') - w(x', Y', i)\right)\delta(x', Y', z') \\
&\quad = (i' - i)\left(\delta(x, Y, z') - \delta(x', Y', z')\right),
\end{aligned}$$

where the last equality follows from the definition of the function w. Therefore, $\Delta_i\left((x, Y), (x', Y')\right)$ is monotonic as i increases from z to z'. Whether it is non-decreasing or non-increasing depends on the sign of $\delta(x, Y, z') - \delta(x', Y', z')$, which is independent of i. □

Combining Lemma 8 with Property 2 shows that each internal node in the higher layers of B undergoes at most one (U2) update between any two consecutive (U1) updates.

4.3 Calculating the function δ

Given two points $X = (x_0, \ldots, x_{d-1})$ and $X' = (x'_0, \ldots, x'_{d-1})$ in $\mathbb{R}^d$, we say that X' is *dominated* by X if $x'_0 \leqslant x_0$ and $x'_i \geqslant x_i$ for all $i \in [d-1]$. The classical *dominance counting* problem asks, for a given query point, how many points in a set of d-dimensional points are dominated by it. This is a special case of the *orthogonal range counting* problem, for which several efficient data structures have been developed with poly-logarithmic query time; see, e.g., [1, 20].

For each $i \in [d-1]$, let $\mathcal{P}_i^+ = U_i \setminus I_i$ denote the subset of $\mathcal{P}^+$ corresponding to the ith branch. Define $\mathcal{Q}_i$ as the set of points $X \in \mathcal{Q}$ for which $\beta(X) = i$, that is, $\mathcal{Q}_i = \mathcal{Q} \cap \mathcal{P}_i^+$. Additionally, let $\mathcal{Q}_i'$ be a multiset obtained by including a point $(0, Y)$ in $\mathcal{Q}_i'$ for each $(x, Y) \in \mathcal{Q} \cap \mathcal{P}_k^+$ with $k \in \text{desc}(i)$.

We can compute $\delta(X)$ for each $X \in \mathcal{P}_i^+$ as the number of points in $\mathcal{Q}_i \cup \mathcal{Q}_i'$ that are dominated by X. Note that the contributions from $\mathcal{Q}_i$ and $\mathcal{Q}_i'$ correspond precisely to the sizes of the sets $\mathcal{D}(X)$ and $\mathcal{D}'(X)$, respectively, used in the definition of $\delta(X)$ in Equation (4).

Lemma 9. *For each point $X \in \mathcal{P}^+$, the function $\delta(X)$ can be computed in poly-logarithmic time.*

Proof. Fix a point $X = (x_0, \ldots, x_{d-1}) \in \mathcal{P}_i^+$ for some $i \in [d-1]$. Recall from Equation (4) that $\delta(X) = |\mathcal{D}(X)| + |\mathcal{D}'(X)|$. According to Equation (2), the set $\mathcal{D}(X)$ consists of all points $X' = (x'_0, \ldots, x'_{d-1}) \in \mathcal{Q}_i$ that are dominated by X. Similarly, by Equation (3), the set $\mathcal{D}'(X)$ contains all points $X' = (x'_0, \ldots, x'_{d-1}) \in \mathcal{Q}_i'$ dominated by X, where each point in $\mathcal{Q}_i'$ is of the form $(0, Y)$ for some $(x, Y) \in \mathcal{Q} \cap \mathcal{P}_k^+$ with $k \in \text{desc}(i)$. The construction of $\mathcal{Q}_i'$ ensures that every point in $\mathcal{D}'(X)$ satisfies the condition $x'_0 = 0 < x_0$, which trivially holds, since $x_0 \geqslant 1$. Therefore, computing $\delta(X)$ reduces to answering two dominance counting queries in d-dimensional space, one over $\mathcal{Q}_i$ and another over $\mathcal{Q}_i'$, which can be done in poly-logarithmic time, after $O(n \operatorname{polylog} n)$ preprocessing time (see, e.g., [20]). □

4.4 The main algorithm

Now, we are ready to provide our main algorithm for solving Problem 3, which in turn solves the maximum edge biclique problem on tree convex bipartite graph as a special case. The pseudocode is provided in Algorithm 1.

Theorem 10. *Algorithm 1 solves Problem 3 in d-dimensional space for any constant $d \geqslant 2$, in $O(n^{d-1} \operatorname{polylog} n)$ time.*

Proof. We first establish the correctness of the algorithm. In Algorithm 1, the (U1) updates are identified in line 5, and the (U3) updates are determined in line 15, both following their definitions. By Corollary 7 and Lemma 8, there is at most one (U2) update between any two consecutive (U1) updates, which are identified in line 12. Furthermore, the (U2) updates that occur before the first or after the last (U1) for all internal nodes are added in line 7. Hence, all updates are correctly identified, and by Lemma 6, the tree is updated accordingly. As such, at each step z, the root of the tree contains the pair (r, S) that maximizes $f(r, S, z)$. Thus, the result returned in line 19 is an optimal solution.

We now analyze the runtime of the algorithm. We can compute $\mathcal{Q}_i$ and $\mathcal{Q}_i'$ from $\mathcal{Q}$ in linear time. Since the dominance counting data structure requires $O(n \operatorname{polylog} n)$ preprocessing time, line 1 of the algorithm executes in $O(n \operatorname{polylog} n)$ time. Once

Algorithm 1 FindMax$(\mathcal{Q}, i)$

```
1: initialize the dominance counting data structure with Q_i ∪ Q'_i
2: construct the tree B, initialized by B_0
3: for each point (x, S, z) ∈ Q do
4:     for each (d − 2)-dimensional vector Y such that Y ⪯ S do
5:         mark ancestors of the leaf corresponding to (x, Y) for (U1) at step z
6: for each node u in B do
7:     mark u for (U2) updates that occur before the first (U1) or after the last one
8: let max ← f(root(B), 0)
9: for each z from 1 to n do
10:    for each node u scheduled for update at step z, in bottom-up order do
11:        if u is marked for a (U1) update then
12:            find the next step z' at which a (U2) update occurs on u
13:            mark u for a (U2) update at step z'
14:        if u is marked for a (U2) update then
15:            mark all ancestors of u for a (U3) update
16:        apply the update to node u
17:    if f(root(B), z) > max then
18:        max ← f(root(B), z)
19: return max
```

initialized, the function δ can be evaluated in polylogarithmic time for any point by Lemma 9. The function f requires computing δ and summing d values for w, and can thus be evaluated in polylogarithmic time.

The binary tree consists of $O(n^{d-1})$ nodes, and initializing it involves computing f for each internal node, resulting in $O(n^{d-1} \operatorname{polylog} n)$ time for line 2. To initialize (U1) updates, the outer loop in line 3 iterates over $O(n)$ elements of $\mathcal{Q}$, while the inner loop in line 4 processes $O(n^{d-2})$ vectors Y. This results in $O(n^{d-1})$ updates of type (U1) at the leaves, and $O(n^{d-1} \log n)$ updates of type (U1) overall. Additionally, line 7 may issue up to two updates per node, contributing another $O(n^{d-1} \log n)$ updates. Therefore, the total time for the initialization phase is $O(n^{d-1} \operatorname{polylog} n)$.

Across the main loop in lines 9 through 18, the total number of (U1) updates is $O(n^{d-1} \log n)$, as established previously. By Corollary 7 and Lemma 8, the number of (U2) updates is also bounded by $O(n^{d-1} \log n)$, since there can be at most one such update between any two consecutive (U1) updates. Moreover, each (U2) update in line 12 can be identified in $O(\log n)$ time via binary search, due to the monotonicity of the gap function established in Corollary 7 and Lemma 8. Each (U2) update subsequently triggers $O(\log n)$ updates of type (U3) on its ancestors. Hence, the total number of updates remains $O(n^{d-1} \log n)$, all of which can be identified and applied in $O(n^{d-1} \operatorname{polylog} n)$ time. Note that applying each update to

a node involves computing only two f values, one for each child. Therefore, the overall time complexity of the algorithm is $O(n^{d-1} \operatorname{polylog} n)$. □

We can now apply Algorithm 1 to solve our original optimization problem, i.e, computing the maximum edge biclique in a tree convex bipartite graph. This yields our main result, stated below.

Theorem 11. *Let $G = (A \cup B, E)$ be a tree convex bipartite graph with n vertices, where the convexity of B is defined over a tree with a constant number d of leaves. Then, a biclique in G with the maximum number of edges can be computed in $O(n^{d-1} \operatorname{polylog} n)$ time.*

Proof. When $d = 2$, the problem reduces to ordinary convex bipartite graphs, which can be solved in $O(n \operatorname{polylog} n)$ time using the algorithm by Nussbaum et al. [30]. For $d \geqslant 3$, Problem 3 for any branch can be solved in $O(n^{d-1} \operatorname{polylog} n)$ time using Algorithm 1, as established in Theorem 10. Since the number of branches is $d - 1$, the overall running time is $O(n^{d-1} \operatorname{polylog} n)$ for any constant d. □

5 Conclusions

In this paper, we revisited the maximum edge biclique problem in tree convex bipartite graphs and presented a faster algorithms that improves upon the previous best algorithm by a factor of $n^3 / \operatorname{polylog} n$.

We can extend our approach to *forest convex bipartite graphs*, where convexity requires that the neighborhood of each vertex induces a connected subgraph within each tree of the underlying forest F, with a total of d leaves. Using a mapping analogous to the one developed in this work, the input can be transformed into a d-dimensional grid, ensuring that any maximum edge biclique corresponds to a subgraph that is connected in every tree and is therefore fully contained within the grid. This reduction enables the extension of our approach to address the maximum edge biclique problem in this generalized setting.

Other classes of convexity, such as *circular convex bipartite graphs*, remain largely unexplored in the context of the maximum edge biclique problem. In these settings, neither efficient algorithms nor formal hardness results are currently known. Investigating the computational complexity and algorithmic possibilities in such classes presents an intriguing direction for future research.

Acknowledgments The authors would like to thank Yeganeh Alimohammadi for her insightful discussions.

References

[1] P. K. Agarwal. Range searching. In *Handbook of discrete and computational geometry*, pages 1057–1092. Chapman and Hall/CRC, 2017.

[2] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16(1):80–109, 1994. doi:10.1006/jagm.1994.1005.

[3] C. Ambühl, M. Mastrolilli, and O. Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM Journal on Computing*, 40(2):567–596, 2011. doi:10.1137/080729256.

[4] A. Bhangale, R. Gandhi, M. T. Hajiaghayi, R. Khandekar, and G. Kortsarz. Bicovering: Covering edges with two small subsets of vertices. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*, volume 55 of *LIPIcs*, pages 6:1–6:12. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.6.

[5] B. S. Biswal, A. Mohapatra, and S. Vipsita. A review on biclustering of gene expression microarray data: algorithms, effective measures and validations. *Int. J. Data Min. Bioinform.*, 21(3):230–268, 2018. doi:10.1504/IJDMB.2018.097683.

[6] F. Bonomo-Braberman, N. Brettell, A. Munaro, and D. Paulusma. Solving problems on generalized convex graphs via mim-width. *Journal of Computer and Systems Sciences*, 140:103493, 2024. doi:10.1016/j.jcss.2023.103493.

[7] 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 Systems Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

[8] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Blocks of hypergraphs - applied to hypergraphs and outerplanarity. In *Proceedings of the 21st International Workshop on Combinatorial Algorithms*, volume 6460 of *LNCS*, pages 201–211. Springer, 2010. doi:10.1007/978-3-642-19222-7_21.

[9] K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. *Journal of Graph Algorithms and Applications*, 15(4):533–549, 2011. doi:10.7155/JGAA.00237.

[10] E. N. Castanho, H. Aidos, and S. C. Madeira. Biclustering data analysis: a comprehensive survey. *Briefings in Bioinformatics*, 25(4):bbae342, 07 2024. doi:10.1093/BIB/BBAE342.

[11] H. Chen, Z. Lei, T. Liu, Z. Tang, C. Wang, and K. Xu. Complexity of domination, hamiltonicity and treewidth for tree convex bipartite graphs. *Journal of Combinatorial Optimization*, 32(1):95–110, 2016. doi:10.1007/S10878-015-9917-3.

[12] H. Chen and T. Liu. Maximum edge bicliques in tree convex bipartite graphs. In *Proceedings of the 11th Frontiers in Algorithmics*, volume 10336 of *LNCS*, pages 47–55. Springer, 2017. doi:10.1007/978-3-319-59605-1_5.

[13] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. R. Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001. doi:10.1006/jagm.2001.1199.

[14] M. Dawande, P. Keskinocak, and S. Tayur. On the biclique problem in bipartite graphs. *Pittsburgh: GSIA Working Paper: Carnegie Mellon University*, 1996.

[15] U. Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 534–543. ACM, 2002. doi:10.1145/509907.509985.

[16] U. Feige and S. Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Technical Report MCS04-04, The Weizmann Institute of Science, 2004.

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

[18] M. Habib, R. M. 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. doi:10.1016/S0304-3975(97)00241-7.

[19] W. Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 43(1):1–16, 2002. doi:10.1006/JAGM.2001.1205.

[20] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proceedings of the 15th International Symposium on Algorithms and Computation*, volume 3341 of *LNCS*, pages 558–568. Springer, 2004. doi:10.1007/978-3-540-30551-4_49.

[21] W. Jiang, T. Liu, C. Wang, and K. Xu. Feedback vertex sets on restricted bipartite graphs. *Theoretical Computer Science*, 507:41–51, 2013. doi:10.1016/j.tcs.2012.12.021.

[22] S. Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006. doi:10.1137/S0097539705447037.

[23] Kusum and A. Pandey. Some new algorithmic results on co-secure domination in graphs. *Theoretical Computer Science*, 992:114451, 2024. doi:10.1016/J.TCS.2024.114451.

[24] M. Lin. Counting dominating sets in some subclasses of bipartite graphs. *Theoretical Computer Science*, 923:337–347, 2022. doi:10.1016/J.TCS.2022.05.021.

[25] M. Lin and C. Chen. Counting independent sets in tree convex bipartite graphs. *Discrete Applied Mathematics*, 218:113–122, 2017. doi:10.1016/J.DAM.2016.08.017.

[26] T. Liu, Z. Lu, and K. Xu. Tractable connected domination for restricted bipartite graphs. *Journal of Combinatorial Optimization*, 29(1):247–256, 2015. doi:10.1007/S10878-014-9729-X.

[27] M. Lu, T. Liu, and K. Xu. Independent domination: Reductions from circular- and triad-convex bipartite graphs to convex bipartite graphs. In *Proceedings of the Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, volume 7924 of *LNCS*, pages 142–152. Springer, 2013. doi:10.1007/978-3-642-38756-2_16.

[28] P. Manurangsi. Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. *Algorithms*, 11(1):10, 2018. doi:10.3390/a11010010.

[29] J. Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, 88(1-3):325–354, 1998. doi:10.1016/S0166-218X(98)00078-X.

[30] D. Nussbaum, S. Pu, J. Sack, T. Uno, and H. Zarrabi-Zadeh. Finding maximum edge bicliques in convex bipartite graphs. *Algorithmica*, 64(2):311–325, 2012. doi:10.1007/s00453-010-9486-x.

[31] B. S. Panda, A. Pandey, J. Chaudhary, P. Dane, and M. Kashyap. Maximum weight induced matching in some subclasses of bipartite graphs. *Journal of Combinatorial Optimization*, 40(3):713–732, 2020. doi:10.1007/S10878-020-00611-2.

[32] K. Paul, A. Sharma, and A. Pandey. Algorithmic results for weak roman domination problem in graphs. *Discrete Applied Mathematics*, 359:278–289, 2024. doi:10.1016/J.DAM.2024.08.007.

[33] R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003. doi:10.1016/S0166-218X(03)00333-0.

[34] B. Pontes, R. Giráldez, and J. S. Aguilar-Ruiz. Biclustering on expression data: A review. *Journal of Biomedical Informatics*, 57:163–180, 2015. doi:10.1016/j.jbi.2015.06.028.

[35] Y. Song, T. Liu, and K. Xu. Independent domination on tree convex bipartite graphs. In *Proceedings of the Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, volume 7285 of *LNCS*, pages 129–138. Springer, 2012. doi:10.1007/978-3-642-29700-7_12.

[36] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(1):S136–S144, 2002. doi:10.1093/bioinformatics/18.suppl_1.s136.

[37] A. Tucker. Matrix characterizations of circular-arc graphs. *Pacific Journal of Mathematics*, 39(2):535–545, 1971.

[38] C. Wang, T. Liu, W. Jiang, and K. Xu. Feedback vertex sets on tree convex bipartite graphs. In *Proceedings of the 6th International Conference on Combinatorial Optimization and Applications*, volume 7402 of *LNCS*, pages 95–102. Springer, 2012. doi:10.1007/978-3-642-31770-5_9.

[39] P. Y. Wang, N. Kitamura, T. Izumi, and T. Masuzawa. Approximation hardness of domination problems on generalized convex graphs. *Theoretical Computer Science*, 1028:115035, 2025. doi:10.1016/J.TCS.2024.115035.