Incremental Maintenance of Maximal Bicliques in a Dynamic Bipartite Graph

Apurba Das, Srikanta Tirthapura, *Senior Member, IEEE,*

Abstract—We consider incremental maintenance of maximal bicliques from a dynamic bipartite graph that changes over time due to the addition of edges. When new edges are added to the graph, we seek to enumerate the change in the set of maximal bicliques, without enumerating the set of maximal bicliques that remain unaffected. The challenge in an efficient algorithm is to enumerate the change without explicitly enumerating the set of all maximal bicliques. In this work, we present (1) Near-tight bounds on the magnitude of change in the set of maximal bicliques of a graph, due to a change in the edge set, and an (2) Incremental algorithm for enumerating the change in the set of maximal bicliques. For the case when a constant number of edges are added to the graph, our algorithm is "change-sensitive", i.e., its time complexity is proportional to the magnitude of change in the set of maximal bicliques. To our knowledge, this is the first incremental algorithm for enumerating maximal bicliques in a dynamic graph, with a provable performance guarantee. Our algorithm is easy to implement, and experimental results show that its performance exceeds that of baseline implementations by orders of magnitude.

Index Terms—Graph Mining, Maximal Biclique Maintenance, Incremental Algorithm, Bipartite Graph

1 Introduction

Graphs are a natural abstraction in representing linked data in many domains such as in social network analysis, computational biology, and web search. Often, these networks are dynamic, where new connections are added and old connections are removed. The area of *dynamic graph mining* focuses on efficient methods for finding and maintaining significant patterns in a dynamic graph. Our work is motivated by applications that require the maintenance of dense substructures from a dynamic graph. Angel et al. [6], propose an algorithm for identifying breaking news stories in real-time through dense subgraph mining from an evolving graph, defined on the co-occurrence of entities within messages in an online social network. Java et al. [17] present methods for detecting communities among users in a microblogging platform through identifying dense structures in an evolving network representing connections among users. A sample of other applications of dense subgraph mining in networks include identification of communities in a social network [15], [23], identification of web communities [14], [34], [18], phylogenetic tree construction [11], [35], [44], communities in bipartite networks [19], genome analysis [30], and closed itemset mining [41], [20].

We focus on *bipartite networks* that can be used to model interactions between two distinct types of entities. For example, a relation between users and news articles in a feed can be modeled as a bipartite network where the users and the news articles are two sets of vertices, and there is an edge between a user and a news article if the user has viewed the article. We consider the problem of *maximal biclique enumeration (MBE)* within an evolving bipartite graph. MBE is a fundamental problem that has been used in detecting communities within large bipartite networks. For instance, the works of Kumar et al. [18] on detecting cyber-communities from the web graph, Murata [29] on identifying user communities from web log data, and Lehmann et al. [19] on community detection in collaboration networks are all based on MBE from an appropriately defined bipartite graph. Another application of MBE is in mining closed itemsets in association rule mining from transactional databases [31]. One approach to closed item-sets is to enumerate maximal bicliques from a bipartite graph representing the transaction database where the different transactions are in one partition and the set of items are in the other partition, with edges connecting a transaction to an item if the item was included in that transaction [20].

The majority of prior work on MBE has focused on a static graph that is assumed to be given in its entirety, and that does not change henceforth. Such methods are not efficient and often are inapplicable for the case when the graph is changing often, such as when more transactions are being added to the database, or when there is more user activity within a network. For example, in community detection, if edges are added to the graph, then it is not clear how to efficiently (re)compute the new communities that emerge, and those communities that are subsumed. In the case of itemset mining, it is a challenge to (re)generate closed itemsets when transactions are added to the database incrementally [41]. Many, if not most, of today's data sources are dynamic and generate data constantly, and it is important to have methods that can handle such dynamic data in an efficient manner.

We consider the incremental MBE problem, of maintaining the set of maximal bicliques in a bipartite graph that is evolving continuously over time due to the addition

• *Das and Tirthapura are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011.*
E-mail: {adas,snt}@iastate.edu.
The authors were partially supported through NSF grants 1527541, 1725702, and 1632116.

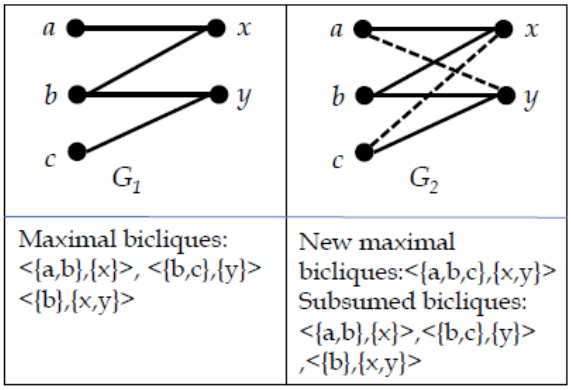


Fig. 1: Change in maximal bicliques when the graph changes from G_1 to G_2 due to the addition of edge set $H = \{\{a, y\}, \{c, x\}\}$. Each maximal biclique in G_1 is subsumed by a larger maximal biclique in G_2, and there is one new maximal biclique in G_2.

of stream of new edges. Let $G = (L, R, E)$ be a simple undirected bipartite graph with its vertex set partitioned into L, R, and edge set $E \subseteq L \times R$. A biclique in G is a bipartition $B = (X, Y)$, $X \subseteq L$, $Y \subseteq R$ such that each vertex in X is connected to each vertex in Y. A biclique B is called a maximal biclique if there is no other biclique B' such that B is a proper subgraph of B'. Let $\mathcal{BC}(G)$ denote the set of all maximal bicliques in G.

Suppose that starting from bipartite graph $G_1 = (L, R, E)$, the state of the graph changes to $G_2 = (L, R, E \cup H)$ due to the addition of a set of new edges H. Let $\Upsilon^{new}(G_1, G_2) = \mathcal{BC}(G_2) \setminus \mathcal{BC}(G_1)$ denote the set of new maximal bicliques that arise in G_2 that were not present in G_1 and $\Upsilon^{del}(G_1, G_2) = \mathcal{BC}(G_1) \setminus \mathcal{BC}(G_2)$ denote the set of maximal bicliques in G_1 that are no longer maximal bicliques in G_2 (henceforth called subsumed bicliques). See Fig. 1 for an example. Let $\Upsilon(G_1, G_2) = \Upsilon^{new}(G_1, G_2) \cup \Upsilon^{del}(G_1, G_2)$ denote the symmetric difference of $\mathcal{BC}(G_1)$ and $\mathcal{BC}(G_2)$. We consider the following questions:

(1) How large can be the size of $\Upsilon(G_1, G_2)$? In particular, can a small change in the set of edges cause a large change in the set of maximal bicliques in the graph?

(2) How can we compute $\Upsilon(G_1, G_2)$ efficiently? Can we quickly compute $\Upsilon(G_1, G_2)$ when $|\Upsilon(G_1, G_2)|$ is small? In short, can we design *change-sensitive algorithms* for enumerating elements of $\Upsilon(G_1, G_2)$, whose time complexity is proportional to the size of change, $|\Upsilon(G_1, G_2)|$?

1.1 Contributions

Magnitude of Change: Let $g(n)$ denote the maximum number of maximal bicliques possible in an n vertex bipartite graph. A result due to Prisner [33] shows that $g(n) \leq 2^{n/2}$, where equality occurs when n is even. We show that the change in the number of maximal bicliques when a single edge is added to the graph can be as large as $3g(n-2) \approx 1.5 \times 2^{n/2}$, which is exponential in the number of vertices in the graph. This shows that the addition of even a single edge to the graph can lead to a large change in the set of maximal bicliques in the graph. We further show that this bound is tight for the case of the addition of a single edge – the largest possible change in the set of maximal bicliques upon adding a single edge is $3g(n-2)$. For the case when more edges can be added to the graph, it is easy to see that the maximum possible change is no larger than $2g(n)$.

Enumeration Algorithm: From our analysis, it is clear that the magnitude of change in the set of maximal bicliques in the graph can be exponential in n in the worst case. On the flip side, the magnitude of change can be as small as 1 – for example, consider the case when a newly arriving edge connects two isolated vertices in the graph. Thus, there is a wide range of values the magnitude of change can take. When the magnitude of change is very large, an algorithm that enumerates the change must inevitably pay a large cost, if only to enumerate the change. On the other hand, when the magnitude of change is small, it will ideally pay a smaller cost. This motivates our search for an algorithm whose computational cost for enumerating the change is proportional to the magnitude of the change in the set of maximal bicliques.

We present an incremental algorithm, `DynamicBC`, for enumerating the change in the set of maximal bicliques when a set of edges H are added to the bipartite graph G. `DynamicBC` has two parts, `NewBC`, for enumerating new maximal bicliques, and `SubBC`, for enumerating subsumed maximal bicliques. When a batch of new edges H of size ρ is added to the graph, the time complexity of `NewBC` for enumerating Υ^{new}, the set of new maximal bicliques, is $O(\Delta^2 \rho |\Upsilon^{new}|)$ where Δ is the maximum degree of the graph after update. The time complexity of `SubBC` for enumerating Υ^{del}, the set of subsumed bicliques, is $O(2^{\rho}|\Upsilon^{new}|)$. Note that when ρ is a constant, the time complexity of enumerating the change is $O(\Delta^2|\Upsilon^{new}|)$, which is linear in the number of bicliques that are output, times a factor related to the size of the graph. To the best of our knowledge, these are the first change-sensitive algorithms for maintaining maximal bicliques in a dynamic graph.

Experimental Evaluation: We present an empirical evaluation of `DynamicBC` on real bipartite graphs with million of nodes and compare our algorithm with baseline approaches. Our results show that the performance of `DynamicBC` is many orders of magnitude faster than directly applying a static algorithm (`BaselineBC`) and many times faster than an improved baseline we devised (`BaselineBC`*). For example, on the `lastfm-song-init` graph, `DynamicBC` took about 93 sec. for computing the change due to addition of 625 batches each of size 100, whereas `BaselineBC` took about $7,920$ sec. and `BaselineBC`* about $1,740$ sec.

1.2 Related Work

Our algorithm belongs to the class of dynamic graph algorithms since it stores the entire graph, and treats changes to the graph in an efficient manner. A graph algorithm is said to be *incremental* if it can (efficiently) handle addition of edges to the graph and *decremental* if it can handle deletions of edges from the graph. For example, the work of Simsiri et al. [37] is an incremental parallel algorithm for graph connectivity that allows addition of edges, and of Thorup [39] is a decremental algorithm for graph connectivity that allows

deletion of edges. A graph algorithm is called fully dynamic if it can handle both addition and deletion of edges, such as the work of Wulff-Nilsen [42] on fully dynamic graph connectivity. Our work thus belongs to the class of incremental graph algorithms, since we consider the addition of edges. Our algorithms also extend to the decremental case in a straightforward manner, since they are able to compute the new as well as subsumed maximal cliques. Our work is also related to the analysis of temporal graphs where edges have timestamps, and arrive (approximately) ordered according to timestamps [25]. Our work cannot however be called a streaming graph algorithm [32], since a streaming graph algorithm typically does not store the entire edge set of the graph. For a problem such as MBE, each edge is important for future structures that may arise, hence, not storing the set of all edges can lead to missing out on enumerating new structures. Hence, the description "dynamic algorithms" better characterizes our work than the description "streaming algorithms".

MBE on a static graph: There has been substantial prior work on enumerating maximal bicliques from a static graph. Alexe et al. [5] present an algorithm for MBE from a static graph based on the consensus method, whose time complexity is proportional to the size of the output (number of maximal bicliques in the graph) - termed as an *output-sensitive algorithm*. Liu et al. [22] present an algorithm for MBE based on depth-first-search (DFS). Damaschke [7] present an algorithm for bipartite graphs with a skewed degree distribution. Gély et al. [13] present an algorithm for MBE through a reduction to maximal clique enumeration (MCE). However, in their work, the number of edges in the graph used for enumeration increases significantly compared to the original graph. Makino & Uno [24] present an algorithm for MBE based on matrix multiplication, which provides the current best time complexity for dense graphs. Eppstein [12] presented a linear time algorithm for MBE when the input graph has bounded arboricity. Other works on sequential algorithms for MBE and MCE on a static graph include [9], [10], [40], [28], and on parallel algorithms include [26], [27], [43], [38]. Li et al. [20] show a correspondence between closed itemsets in a transactional database and maximal bicliques in an appropriately defined graph.

Dense Structures from Dynamic Graphs: There have been some prior works related to maintenance of dense structures similar to maximal bicliques in dynamic graphs. Kumar et al. [18] define an (i, j)-core which is a biclique with i vertices in one partition and j vertices in another partition, and present a dynamic algorithm for extracting non-overlapping sets of (i, j)-cores for interesting communities. Some other works on maintaining dense structures in dynamic graph include maintenance of k-cores [36], [21], k-truss [16], maximal clique [8] etc.

Roadmap: The remaining sections are organized as follows. We present definitions and preliminaries in Section 2. Then we describe our algorithms in Section 3, results on the size of change in the set of maximal bicliques in Section 4, and experimental results in Section 5.

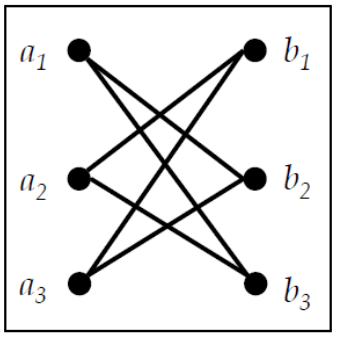


Fig. 2: Cocktail-party graph on 6 vertices $CP(3)$

2 Preliminaries

Let $V(G)$ denote the set of vertices of G and $E(G)$ the set of edges in G. Let n and m denote the number of vertices and number of edges in G respectively. Let $\Gamma_G(u)$ denote the set of vertices adjacent to vertex u in G. If the graph G is clear from the context, we use $\Gamma(u)$ to mean $\Gamma_G(u)$. For an edge $e = (u, v) \in E(G)$, let $G - e$ denote the graph after deleting $e \in E(G)$ from G and $G + e$ denote the graph after adding $e \notin E(G)$ to G. For a set of edges H, let $G + H$ ($G - H$) denote the graph obtained after adding (deleting) H to (from) $E(G)$. Similarly, for a vertex $v \notin V(G)$, let $G+v$ denote the graph after adding v to G and for a vertex $v \in V(G)$, let $G - v$ denote the graph after deleting v and all its adjacent edges from $E(G)$. Let $\Delta(G)$ denote the maximum degree of a vertex in G and $\delta(G)$ the minimum degree of a vertex in G.

Definition 1 (Change-Sensitive Algorithm)**.** *An algorithm for a property P on a dynamic graph is said to be change-sensitive if the time complexity of enumerating the change in P is linear in the magnitude of change (in P), and polynomial in the size of the input graph and the size of change in the set of edges.*

We note that this usage of "change-sensitive" is consistent with prior usage of the term "output-sensitive" in the literature. Algorithms for enumerating maximal bicliques from a static graph are called "output-sensitive" if their runtime is linear in the size of the output (the total number of maximal bicliques), times a factor polynomial in the number of vertices and the number of edges in the graph. For instance, the output-sensitive algorithm for MBE due to Alexe et al. [5] takes time $O(n^3\beta)$, where n is the number of vertices in the graph and β is the number of maximal bicliques, which is the relevant output.

Results for a static graph. In [33], Prisner presented the following result on the number of maximal bicliques in a bipartite graph with n vertices. Let $CP(k)$ denotes the *cocktail-party* graph which is a bipartite graph with k vertices in each partition where $V(CP(k)) = \{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}$ and $E(CP(k)) = \{(a_i, b_p) : i \neq p\}$ [33]. See Figure 2 for an example.

Theorem 1 (Theorem 2.1 [33])**.** *Every bipartite graph with n vertices contains at most $2^{\frac{n}{2}} \approx 1.41^n$ maximal bicliques, and the only extremal (maximal) bipartite graphs are the graphs $CP(k)$.*

As a subroutine, we use an algorithm for enumerating maximal bicliques from a static undirected graph, whose runtime is proportional to the number of maximal bicliques. There are a few such algorithms [5], [22], [45]. We use the following result due to Liu et al. [22] as it provides the current best time complexity.

Theorem 2 (Liu et al., [22]). *For a graph G with n vertices, m edges, maximum degree Δ, and number of maximal bicliques μ, there is an algorithm* `MineLMBC` *for enumerating maximal bicliques in G with time complexity $O(n\Delta\mu)$ and space complexity $O(m + \Delta^2)$.*

`MineLMBC` is an algorithm for enumerating maximal bicliques of a static graph $G = (V, E)$ that is based on depth-first-search. It takes as input the graph G and the size threshold s. The algorithm enumerates all maximal bicliques of G with size of each partition at least s. Clearly, by setting $s = 1$, the algorithm enumerates all maximal bicliques of G.

3 Algorithms for Maximal Bicliques

For graph G and set of edges H, we use Υ^{new} to mean $\Upsilon^{new}(G, G + H)$, and Υ^{del} to mean $\Upsilon^{del}(G, G + H)$. Before presenting our change-sensitive algorithm for maximal bicliques, we first consider two baseline approaches for the problem.

3.1 Baseline Algorithms for Maximal Bicliques

First we consider a straightforward approach for maintaining maximal bicliques using a current state-of-the-art algorithm for static graphs. This algorithm, which we call as `BaselineBC`, works by enumerating $\mathcal{BC}(G + H)$, the set of all maximal bicliques in $(G+H)$ once G is updated with a set of new edges H. It then outputs the symmetric difference between $\mathcal{BC}(G)$ (maintained in memory) and $\mathcal{BC}(G + H)$.

We next present another baseline `BaselineBC`*, which is better than `BaselineBC`. The idea in `BaselineBC`* is to focus on the portion of the graph where changes occur. Let V_H denote the set of all vertices in G that are incident to at least one edge in H. For enumerating new maximal bicliques, we note that it is sufficient to consider the subgraph G_H of $G + H$ that is induced by V_H and the vertices in $\cup_{v \in V_H} \Gamma_{G+H}(v)$. `BaselineBC`* enumerates all maximal bicliques in G_H using a state-of-the-art algorithm for static graphs. Each biclique b thus generated is a new maximal biclique if b contains at least an edge from H. For enumerating subsumed bicliques, we note each subsumed maximal biclique b' in G is a subgraph of at least one new maximal biclique b, and must also be contained in $b - H$. Thus, subsumed maximal bicliques are enumerated by considering each new maximal b, and enumerating maximal bicliques in $b - H$. If a biclique thus enumerated is present in $\mathcal{BC}(G)$, it is output as a subsumed biclique.

We can expect `BaselineBC`* to do much better than `BaselineBC`. Still `BaselineBC`* it is not change-sensitive, because it may, in the process of enumerating new maximal bicliques, generate bicliques of G that remain maximal in $G + H$. For example, see Fig. 3. We next present algorithms that carefully avoid enumerating any maximal biclique of G that remains maximal in $G + H$.

3.2 Change-Sensitive Algorithm `DynamicBC`

Our change-sensitive algorithm, `DynamicBC`, has two parts: (1) Algorithm `NewBC` for enumerating new maximal bicliques, described in Section 3.3 and (2) Algorithm `SubBC` for enumerating subsumed bicliques, described in Section 3.4. The main result on the time complexity of `DynamicBC` is

Algorithm 1: `DynamicBC`$(G, H, \mathcal{BC}(G))$

Input: G - Input bipartite graph, H - Edges being added to G, $\mathcal{BC}(G)$ - set of maximal bicliques of G

Output: Υ : the union of set of new maximal bicliques and subsumed bicliques

1 $\Upsilon^{new} \leftarrow$ `NewBC`(G, H)
2 $\Upsilon^{del} \leftarrow$ `SubBC`$(G, H, \mathcal{BC}(G), \Upsilon^{new})$
3 $\Upsilon \leftarrow \Upsilon^{new} \cup \Upsilon^{del}$

summarized in the following theorem.

Theorem 3. `DynamicBC` *enumerates the change in the set of maximal bicliques, with time complexity $O(\Delta^2\rho|\Upsilon^{new}| + 2^\rho|\Upsilon^{new}|)$ where Δ is the maximum degree of a vertex in $G + H$ and ρ is the size of H, the set of newly added edges.*

We note that if ρ is constant, the time complexity of enumerating the change is $O(\Delta^2|\Upsilon^{new}|)$. Thus we have the following observation.

Observation 1. `DynamicBC` *is a change-sensitive algorithm for MBE, when the number of edges added, ρ is a constant.*

3.3 Enumerating New Maximal Bicliques

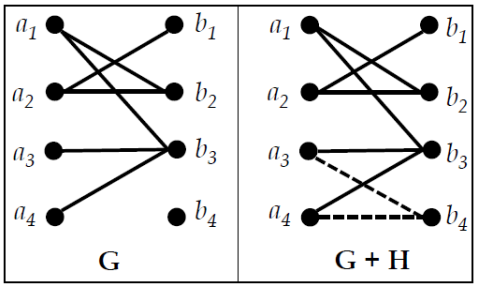


Fig. 3: The original graph G has 4 maximal bicliques. When new edges in H (in dotted line) are added to G, all maximal bicliques in G remain maximal in $G + H$ and only one maximal biclique is newly formed ($< \{a_3, a_4\}, \{b_3, b_4\} >$).

In our algorithm, we require that each maximal biclique enumerated by `NewBC` to contain at least one edge from H, thus forcing it to be a new maximal biclique. Let G' denote the graph $G + H$. For each new edge $e \in H$, let $\mathcal{BC}'(e)$ denote the set of maximal bicliques in G' containing edge e.

Lemma 1. $\Upsilon^{new} = \cup_{e \in H} \mathcal{BC}'(e)$.

Proof. Each biclique in Υ^{new} must contain at least one edge from H. To see this, consider a biclique $b \in \Upsilon^{new}$. If b did not contain an edge from H, then b is also a maximal biclique in G, and hence cannot belong to Υ^{new}. Hence, $b \in \mathcal{BC}'(e)$ for some edge $e \in H$, and $b \in \cup_{e \in H} \mathcal{BC}'(e)$. This shows that $\Upsilon^{new} \subseteq \cup_{e \in H} \mathcal{BC}'(e)$.

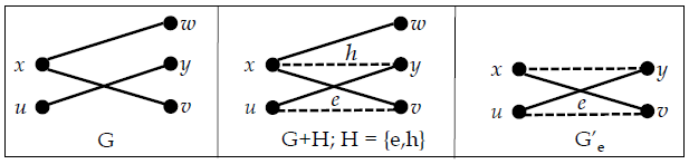


Fig. 4: Construction of G'_e from $G' = G + H$ when a set of new edges $H = \{e, h\}$ is added to G. $A = \Gamma_{G'}(v) = \{u, x\}$ and $B = \Gamma_{G'}(u) = \{v, y\}$.

Next consider a biclique $b \in \cup_{e \in H} \mathcal{BC}'(e)$. It must be the case that $b \in \mathcal{BC}'(h)$ for some h in H. Thus b is a maximal biclique in $G + H$. Since b contains edge $h \in H$, b cannot be a biclique in G. Thus $b \in \Upsilon^{new}$. This shows that $\cup_{e \in H} \mathcal{BC}'(e) \subseteq \Upsilon^{new}$. □

Next, for each edge $e = (u, v) \in H$, we present an efficient way to enumerate all bicliques in $\mathcal{BC}'(e)$ through enumerating maximal bicliques in a specific subgraph G'_e of G', constructed as follows. Let $A = \Gamma_{G'}(u)$ and $B = \Gamma_{G'}(v)$. Then $G'_e = (A, B, E')$ is a subgraph of G' induced by vertices in A and B, and all edges between these sets of vertices. See Fig. 4 for an example of the construction of G'_e.

Lemma 2. *For each $e \in H$, $\mathcal{BC}'(e) = \mathcal{BC}(G'_e)$*

Proof. First we show that $\mathcal{BC}'(e) \subseteq \mathcal{BC}(G'_e)$. Consider a biclique $b = (X, Y)$ in $\mathcal{BC}'(e)$. Let $e = (u, v)$. Here b contains both u and v. Suppose that $u \in X$ and $v \in Y$. According to the construction G'_e contains all the vertices adjacent to u and all the vertices adjacent to v. And in b, all the vertices in X are connected to all the vertices in Y. Hence, b is a biclique in G'_e. Also, b is a maximal biclique in G', and G'_e is an induced subgraph of G' which contains all the vertices of b. Hence, b is a maximal biclique in G'_e.

Next we show that $\mathcal{BC}(G'_e) \subseteq \mathcal{BC}'(e)$. Consider a biclique $b' = (X', Y')$ in $\mathcal{BC}(G'_e)$. Clearly, b' contains e as it contains both u and v and b' is a maximal biclique in G'_e. Hence, b' is also a biclique in G' that contains e. Now we prove that b' is also maximal in G'. Suppose not, that there is a vertex $w \in V(G')$ such that b' can be extended with w. Then, as per the construction of G'_e, $w \in V(G'_e)$ since w must be adjacent to either u or v. Then, b' is not maximal in G'_e. This is a contradiction. Hence, b' is also maximal in G'. Therefore, $b' \in \mathcal{BC}'(e)$. □

Based on the above observation, we present our change-sensitive algorithm `NewBC` (Algorithm 2). We use an output-sensitive algorithm for a static graph `MineLMBC` for enumerating maximal bicliques from G'_e. Note that typically, G'_e is much smaller than G' since it is localized to edge e, and hence enumerating all maximal bicliques from G'_e should be relatively inexpensive.

Theorem 4. `NewBC` *enumerates the set of all new bicliques arising from the addition of H in time $O(\Delta^2 \rho |\Upsilon^{new}|)$ where Δ is the maximum degree of a vertex in G' and ρ is the size of H. The space complexity is $O(|E(G')| + \Delta^2)$.*

Proof. First we consider correctness of the algorithm. From Lemma 1 and Lemma 2, we know that Υ^{new} is enumerated by enumerating $\mathcal{BC}(G'_e)$ for every $e \in H$. Our algorithm does this exactly, and uses the `MineLMBC` algorithm for enumerating $\mathcal{BC}(G'_e)$. For the runtime, consider that the algorithm iterates over each edge e in H. In each iteration, it constructs a graph G'_e and runs `MineLMBC`(G'_e). Note that the number of vertices in G'_e is no more than 2Δ, since it is the size of the union of the edge neighborhoods of one of the ρ edges in G'. The set of maximal bicliques generated in each iteration is a subset of Υ^{new}, therefore the number of maximal bicliques generated from each iteration is no more than $|\Upsilon^{new}|$. From Theorem 2, we have that the runtime of each iteration is $O(\Delta^2 |\Upsilon^{new}|)$. Since there are ρ edges in H, the result on runtime follows. For the space complexity, we note that the algorithm does not store the set of new bicliques in memory at any point. The space required to construct G'_e is linear in the size of G'. From Theorem 2, the total space requirement is $O(|E(G')| + \Delta^2)$. □

Algorithm 2: `NewBC`(G, H)

Input: G - Input bipartite graph, H - Edges being added to G
Output: bicliques in Υ^{new}, each biclique output once

```
Consider edges of H in an arbitrary order e_1, e_2, ..., e_ρ
G' ← G + H
for i = 1 ... ρ do
    e ← e_i = (u, v)
    G'_e ← a subgraph of G' induced by Γ_G'(u) ∪ Γ_G'(v)
    Generate bicliques of G'_e using MineLMBC. Let B denote the set of the generated bicliques.
    for b ∈ B do
        if b does not contain an edge e_j for j < i then
            Add b to Υ^new
return Υ^new
```

3.4 Enumerating Subsumed Maximal Bicliques

We now consider enumerating $\mathcal{BC}(G) \setminus \mathcal{BC}(G')$ where $G' = G + H$. Suppose a new maximal biclique b of G' subsumed a maximal biclique b' of G. Note that b' is also a maximal biclique in $b - H$. One approach is to enumerate all maximal bicliques in $b - H$ and then check which among them is maximal in G. However, checking maximality of a biclique is a costly operation in itself, since we need to consider the neighborhood of every vertex in the biclique. Another idea is to store the bicliques of the graph explicitly and see which among the generated bicliques are contained in the set of maximal bicliques of G. This is not desirable either, since large amount of memory is required to store the set of all maximal bicliques of G.

We consider a more efficient approach, of storing the signatures of the maximal bicliques instead of storing the bicliques themselves. We then enumerate all maximal bicliques in $b - H$ and for each biclique thus generated, we compare the signature of the generated biclique with the signatures of the bicliques stored. An algorithm following this idea is presented in Algorithm 3. This reduces the memory requirement. We use a standard hash function (the

64 bit murmur hash [1]). For computing the signature of a biclique, first we represent the biclique in a canonical form (vertices in first partition represented in lexicographic order followed by vertices in another partition represented in lexicographic order). Then we convert the string into bytes, and apply the hash function to derive the signature. By storing hash signatures instead of maximal bicliques, we are able to quickly check whether a maximal biclique from $b - H$ is contained in the set of maximal bicliques of G by comparing their hash values. We also pay a lower memory cost, when compared with storing all bicliques.

With the approach of storing hash signatures of bicliques, there is a small probability of a hash collision, i.e. the case when two bicliques A and B are unequal, but their hash values are equal. The effect of a collision is a false positive – our algorithm may incorrectly conclude that a biclique is a subsumed biclique where as it is not. However, this is a very unlikely event with the use of 64 bit signatures, where the chances of two unequal strings having the same hash value is extremely small. In our experiments, the set of bicliques that were enumerated by our algorithm always matches the set of subsumed bicliques. Note that we can always double check each such biclique by explicitly checking if this is maximal in G, to avoid a chance of a false positive. We did not do this in our implementation since the chance of a false positive is so small.

Now we prove that Algorithm 3 enumerates all maximal bicliques of $b - H$.

Lemma 3. *In Algorithm 3, for each $b \in \Upsilon^{new}$, S after Line 14 contains all maximal bicliques in $b - H$.*

Proof. First observe that, removing H from b is equivalent to removing those edges in H which are present in b. Hence, computing maximal bicliques in $b-H$ reduces to computing maximal bicliques in $b - H_1$ where H_1 is the set of all edges in H which are present in b. We use induction on the number of edges k in H_1. Consider the base case, when $k = 1$. H_1 contains a single edge $e_1 = \{u, v\}$. Clearly, $b - H_1$ has two maximal bicliques $b \setminus \{u\}$ and $b \setminus \{v\}$. Suppose, that the set H_1 is of size k. Our inductive hypothesis is that all maximal bicliques in $b - H_1$ are enumerated. Consider $H_1' = \{e_1, e_2, ..., e_k, e_{k+1}\}$ with $k + 1$ edges. Now each maximal biclique b' in $b - H_1$ either remains maximal within $b - H_1'$ (if at least one endpoint of e_{k+1} is not in b') or generates two maximal bicliques in $b - H_1'$ (if both endpoints of e_{k+1} are in b'). Thus, for each $b \in \Upsilon^{new}$, S after Line 14 contains all maximal bicliques within $b - H$. □

We now show that the above algorithm is a change-sensitive algorithm for enumerating all elements of Υ^{del} when the number of edges ρ in H is constant.

Theorem 5. *Algorithm 3 enumerates all bicliques in $\Upsilon^{del} = \mathcal{BC}(G) - \mathcal{BC}(G + H)$ using time $O(2^{\rho}|\Upsilon^{new}|)$ where ρ is the number of edges in H. The space complexity of the algorithm is $O(|E(G')| + |V(G')| + \Delta^2 + |\mathcal{BC}(G)|)$.*

Proof. We first show that every biclique b' enumerated by the algorithm is indeed a biclique in Υ^{del}. Note that b' is a maximal biclique in G, due to explicitly checking the condition. Further, b' is not a maximal biclique in $G+H$, since it is a proper subgraph of b, a maximal biclique in $G + H$. Next, we show that all bicliques in Υ^{del} are enumerated. Consider any subsumed biclique $b' \in \Upsilon^{del}$. It must be contained within $b \setminus H$, where b is a maximal biclique within Υ^{new}. Moreover, b' will be a maximal biclique within $b \setminus H$, and will be enumerated by the algorithm according to Lemma 3.

For the time complexity we show that for any $b \in \Upsilon^{new}$, the maximum number of maximal bicliques in $b - H$ is 2^{ρ} using induction on ρ. Suppose $\rho = 1$ so that H contains a single edge, say $e_1 = (u, v)$. Then, $b - H$ has two maximal bicliques, $b \setminus \{u\}$ and $b \setminus \{v\}$, proving the base case. Suppose that for any set H of size k, it was true that $b - H$ has no more than 2^k maximal bicliques. Consider a set $H'' = \{e_1, e_2, \ldots, e_{k+1}\}$ with $k + 1$ edges. Let $H' = \{e_1, e_2, \ldots, e_k\}$. Subgraph $b - H''$ is obtained from $b - H'$ by deleting a single edge e_{k+1}. By induction, we have that $b - H'$ has no more than 2^k maximal bicliques. Each maximal biclique b' in $b - H'$ either remains a maximal biclique within $b - H''$ (if at least one endpoint of e_{k+1} is not in b'), or leads to two maximal bicliques in $b - H''$(if endpoints of e_{k+1} are in different bipartition of b'). Hence, the number of maximal bicliques in $b - H''$ is no more than 2^{k+1}, completing the inductive step. Following this, for each biclique $b \in \Upsilon^{new}$, we need to check for maximality for no more than 2^{ρ} bicliques in G. This checking can be performed by checking whether each such generated biclique in contained in the set $\mathcal{BC}(G)$ and for each biclique,

1. https://sites.google.com/site/murmurhash/

Algorithm 3: SubBC$(G, H, BC, \Upsilon^{new})$

Input: G - Input bipartite graph
H - Edge set being added to G
BC - Set of maximal bicliques in G
Υ^{new} - set of new maximal bicliques in $G + H$
Output: All bicliques in $\Upsilon^{del} = \mathcal{BC}(G) \setminus \mathcal{BC}(G + H)$

1 $\Upsilon^{del} \leftarrow \emptyset$
2 **for** $b \in \Upsilon^{new}$ **do**
3 $S \leftarrow \{b\}$
4 **for** $e = (u, v) \in E(b) \cap H$ **do**
5 $S' \leftarrow \phi$
6 **for** $b' \in S$ **do**
7 **if** $e \in E(b')$ **then**
8 $b_1 = b' \setminus \{u\}$; $b_2 = b' \setminus \{v\}$
9 $S' \leftarrow S' \cup \{b_1, b_2\}$
10 **else**
11 $S' \leftarrow S' \cup b'$
12 /* S' contains all the maximal bicliques in $b - \{e_1, e_2, ..., e_k\}$ where $\{e_1, e_2, ..., e_k\} \subseteq E(b) \cap H$ are considered so far. */
13
14 $S \leftarrow S'$
15 **for** $b' \in S$ **do**
16 **if** $b' \in BC$ **then**
17 Add b' to Υ^{del}
18 $BC \leftarrow BC \setminus b'$
19 **return** Υ^{del}

this can be done in constant time.

For the space bound, we first note that in Algorithm 3, enumerating maximal bicliques within $b - H$ consumes space $O(|E(G')| + \Delta^2)$, and checking for maximality can be done in space linear in size of G. However, for storing the maximal bicliques in G takes $O(|\mathcal{BC}(G)|)$ space. Hence, for these operations, the overall space-cost for each $b \in \Upsilon^{new}$ is $O(|E(G')| + |V(G')| + \Delta^2 + |\mathcal{BC}(G)|)$. The only remaining space cost is the size of Υ^{new}, which can be large. Note that, the algorithm only iterates through Υ^{new} in a single pass. If elements of Υ^{new} are provided as a stream from the output of an algorithm such as `NewBC`, then they do not need to be stored within a container, so that the memory cost of receiving Υ^{new} is reduced to the cost of storing a single maximal biclique within Υ^{new} at a time. □

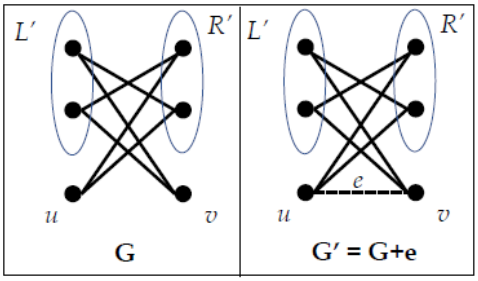


Fig. 5: Construction showing the changes in the set of maximal bicliques when a new edge is added. G is in the left on $n = 6$ vertices. G'' consists of vertices in L' and R' and edges among them to make it a cocktail-party graph. G' in the right is obtained by adding edge $e = (u, v)$ to G.

4 Magnitude of Change in Bicliques

We consider the maximum change in the set of maximal bicliques when a set of edges is added to the bipartite graph. Let $\lambda(n)$ denote the maximum size of $\Upsilon(G, G + H)$ taken over all n vertex bipartite graphs G and edge sets H. We derive the following upper bound on the maximum size of $\Upsilon(G, G + H)$ in the following Lemma:

Lemma 4. $\lambda(n) \leq 2g(n)$.

Proof. Note that, for any bipartite graph G with n vertices and for any new edge set H it must be true that $|\mathcal{BC}(G)| \leq g(n)$ and $|\mathcal{BC}(G + H)| \leq g(n)$. Since $|\Upsilon^{new}(G, G + H)| \leq |\mathcal{BC}(G + H)|$ and $|\Upsilon^{del}(G, G + H)| \leq |\mathcal{BC}(G)|$, it follows that $|\Upsilon(G, G + H)| \leq |\mathcal{BC}(G + H)| + |\mathcal{BC}(G)| \leq 2g(n)$. □

Next we analyze the upper bound of $|\Upsilon(G, G+e)|$ in the following when an edge $e \notin E(G)$ is added to G.

Theorem 6. *For an integer $n \geq 2$, a bipartite graph $G = (L, R, E)$ with n vertices, and any edge $e = (u, v) \notin E(G), u \in L, v \in R$, the maximum size of $\Upsilon(G, G + e)$ is $3g(n - 2)$, and for each even n, there exists a bipartite graph that achieves this bound.*

We prove this theorem in the following two lemmas. In Lemma 5 we prove that the size of $\Upsilon(G, G + e)$ can be as large as $3g(n - 2)$ and in Lemma 7 we prove that the size of $\Upsilon(G, G + e)$ is at most $3g(n - 2)$.

Lemma 5. *For any even integer $n > 2$ there exists a bipartite graph G on n vertices and an edge $e = (u, v) \notin E(G)$ such that $|\Upsilon(G, G + e)| = 3g(n - 2)$.*

Proof. We use proof by construction. Consider bipartite graph $G = (L, R, E)$ constructed on vertex set $L \cup R$ with n vertices such that $|L| = |R| = n/2$. Let $u \in L$ and $v \in R$ be two vertices and let $L' = L \setminus \{u\}$ and $R' = R \setminus \{v\}$. Let G'' denote the induced subgraph of G on vertex sets L' and R'. In our construction, G'' is $CP(\frac{n}{2} - 1)$. In graph G, in addition to the edges in G'', we add an edge from each vertex in R' to u and an edge from each vertex in L' to v. We add edge $e = (u, v)$ to G to get graph $G' = G + e$ (see Fig. 5 for construction). We claim that the size of $\Upsilon(G, G')$ is $3g(n - 2)$.

First, we note that the total number of maximal bicliques in G is $2g(n-2)$. Each maximal biclique in G contains either vertex u or v, but not both. The number of maximal bicliques that contain vertex u is $g(n-2)$, since each maximal biclique in G'' leads to a maximal biclique in G by adding u. Similarly, the number of maximal bicliques in G that contains v is $g(n - 2)$, leading to a total of $2g(n - 2)$ maximal bicliques in G.

Next, we note that the total number of maximal bicliques in G' is $g(n-2)$. To see this, note that each maximal biclique in G' contains both vertices u and v. Further, for each maximal biclique in G'', we get a corresponding maximal biclique in G' by adding vertices u and v. Hence the number of maximal bicliques in G' equals the number of maximal bicliques in G'', which is $g(n - 2)$.

No maximal biclique in $\mathcal{BC}(G)$ contains both u and v, while every maximal biclique in G' contains both u and v. Hence, $\mathcal{BC}(G)$ and $\mathcal{BC}(G')$ are disjoint sets, and $|\Upsilon(G, G')| = |\mathcal{BC}(G)| + |\mathcal{BC}(G')| = 3g(n - 2)$. □

Now we will prove a few results that we will use in proving Lemma 7.

Lemma 6. *If $e = (u, v)$ is added to G, each biclique $b \in \mathcal{BC}(G) - \mathcal{BC}(G + e)$ contains either u or v.*

Proof. Proof by contradiction. Suppose there is maximal biclique $b = (b_1, b_2)$ in $\mathcal{BC}(G) - \mathcal{BC}(G + e)$ that contain neither u nor v. Then, b must be maximal biclique in G. Since b is not a maximal biclique in $G + e$, b is contained in another maximal biclique $b' = (b'_1, b'_2)$ in $G+e$. Note that b' must contain edge $e = (u, v)$, and hence, both vertices u and v. Since b' is a biclique, every vertex in b'_2 is connected to u in G'. Hence, every vertex in b_2 is connected to u even in G. Therefore, $b \cup \{u\}$ is a biclique in G, and b is not maximal in G, contradicting our assumption. □

Observation 2. *For a bipartite graph $G = (L, R, E)$ and a vertex $u \in V(G)$, the number of maximal bicliques that contains u is at most $g(n - 1)$.*

Proof. Suppose, $u \in L$. Then each maximal biclique b in G that contains u, corresponds to a unique maximal biclique in $G - \{u\}$. Such maximal bicliques can be derived from b by deleting u from b. As the maximum number of maximal bicliques in $G - \{u\}$ is $g(n - 1)$, the maximum number of maximal bicliques in G can be no more than $g(n - 1)$. □

Observation 3. *The number of maximal bicliques containing a specific edge (u, v) is at most $g(n-2)$.*

Proof. Consider an edge $(u, v) \in E(G)$. Let vertex set $V' = (\Gamma_G(u) \cup \Gamma_G(v)) - \{u, v\}$, and let G' be the subgraph of G induced by V'. Each maximal biclique b in G that contains edge (u, v) corresponds to a unique maximal biclique in G' by simply deleting vertices u and v from b. Also, each maximal biclique b' in G' corresponds to a unique maximal biclique in G that contains (u, v) by adding vertices u and v to b'. Thus, there is a bijection between the maximal bicliques in G' and the set of maximal bicliques in G that contains edge (u, v). The number of maximal bicliques in G' can be at most $g(n-2)$ since G' has no more than $(n-2)$ vertices, completing the proof. □

Lemma 7. *For a bipartite graph $G = (L, R, E)$ on n vertices and edge $e = (u, v) \notin E(G)$, the size of $\Upsilon(G, G+e)$ can be no larger than $3g(n-2)$.*

Proof. Proof by contradiction. Suppose there exists a bipartite graph $G = (L, R, E)$ and edge $e \notin E(G)$ such that $|\Upsilon(G, G+e)| > 3g(n-2)$. Then either $|\mathcal{BC}(G+e) - \mathcal{BC}(G)| > g(n-2)$ or $|\mathcal{BC}(G) - \mathcal{BC}(G+e)| > 2g(n-2)$.

Case 1: $|\mathcal{BC}(G+e) - \mathcal{BC}(G)| > g(n-2)$: This means that total number of new maximal bicliques formed due to addition of edge e is larger than $g(n-2)$. Note that each new maximal biclique formed due to addition of e must contain e. From Observation 3, the total number of maximal bicliques in an n vertex bipartite graph containing a specific edge can be at most $g(n-2)$. Thus, the number of new maximal bicliques after adding edge e is at most $g(n-2)$, contradicting our assumption.

Case 2: $|\mathcal{BC}(G) - \mathcal{BC}(G+e)| > 2g(n-2)$: Using Lemma 6, each maximal biclique $b \in \mathcal{BC}(G) - \mathcal{BC}(G+e)$ must contain either u or v, but not both. Suppose that b contains u but not v. Then, b must be a maximal biclique in $G - v$. Using Observation 2, we see that the number of maximal bicliques in $G - v$ that contains a specific vertex u is no more than $g(n-2)$. In a similar way, the number of possible maximal bicliques that contain v is at most $g(n-2)$. Therefore, the total number of maximal bicliques in $\mathcal{BC}(G) - \mathcal{BC}(G+e)$ is at most $2g(n-2)$, contradicting our assumption. □

Combining Lemma 4, Theorem 6 and using the fact that $3g(n-2) = 1.5g(n)$ for even n, we obtain the following when n is even:

Theorem 7. $1.5g(n) \leq \lambda(n) \leq 2g(n)$

5 Experimental Evaluation

In this section, we present results of an experimental evaluation of our algorithms.

5.1 Data

We consider the following real-world bipartite graphs in our experiments. A summary of the datasets is presented in Table 1. We collect all the datasets from KONECT - The Koblenz Network Collection [2]. In the `epinions-rating` [1] graph, vertices consist of users in one partition and products in another partition. There is an edge between a user and a product if the user rated that product. In the `lastfm-song` [2] graph, vertices consist of users in one partition and the songs in another partition. When a user listens to a song an edge connect the user with that song. In the `movielens-10M` [3] graph, vertices consist of users in one partition and movies in another partition. There is an edge between a user and a movie if the user rated that movie. In the `wiktionary` [4] graph, vertices consist of users and pages from English Wiktionary. There is an edge between a user and a page if that user edited the page. Each bipartite graph has timestamps on their edges. We converted each bipartite graph into a simple undirected bipartite graph by ignoring edge directions, and considered the earliest creation time of an edge as the timestamp, if there are multiple edges in the original graph.

We create the initial graphs (`epinions-rating-init`, `lastfm-song-init`, `movielens-10M-init`, and `wiktionary-init`) by removing all the edges from the original graphs and present the edge stream in the increasing order of their time-stamps. In Table 1, column `Edges (start)` represents the number of edges in the initial graph and column `Edges (stop)` represents the total number of edges inserted until we stop the experiment. For each input graph we run each algorithm upto 2 hours.

2. http://konect.uni-koblenz.de/

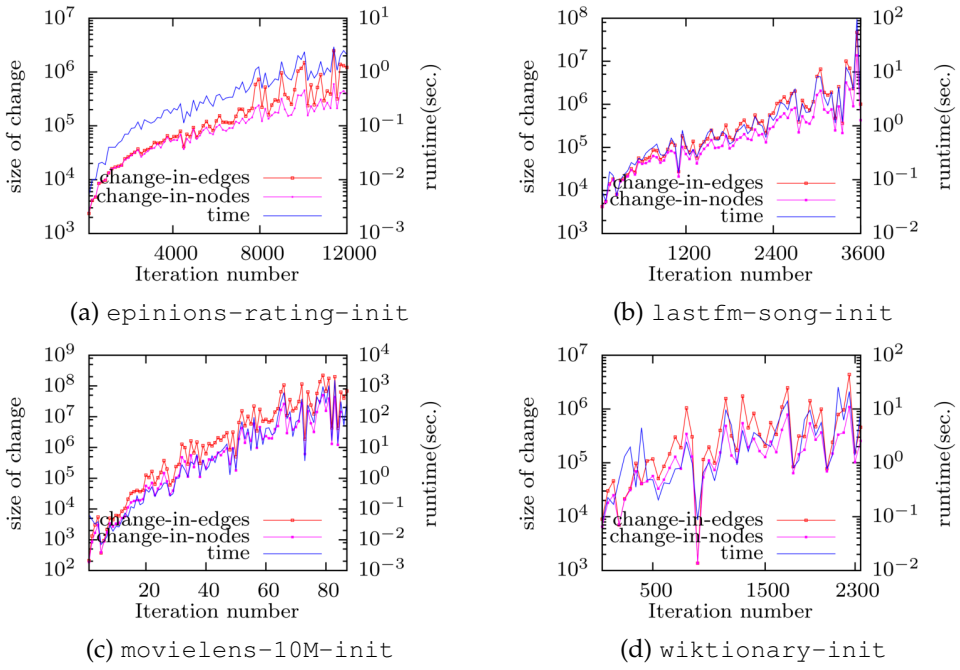
5.2 Experimental Setup and Implementation Details

We implemented all algorithms in Java on a 64-bit Intel(R) Xeon(R) CPU clocked at 3.10 Ghz and 8G DDR3 RAM with 6G heap memory space. Unless otherwise specified, each batch consists of 100 edges and size threshold $s = 1$, where the size threshold refers to the minimum size of each partition of a maximal biclique. We report the median of 3 runs for each input graph.

Metrics: We evaluate our algorithms using the following metrics: (1) computation time for new maximal bicliques and subsumed bicliques when a set of edges are added, (2) memory consumption, that is the main memory used by the algorithm for storing the graph, and other data structures used by the algorithm, (3) cumulative time, that is the total computation time from the initial graph till we stop the experiment with different batch sizes, and (4) change-sensitiveness, the relation of the total computation time to the size of change. We measure the size of change as the sum of the total number of edges in the new maximal bicliques and the subsumed bicliques (change-in-edges) as well as the sum of the total number of nodes (change-in-nodes) and

5.3 Discussion of Results

Comparison with Baseline Algorithms: We compare the performance of `DynamicBC` with baseline algorithms `BaselineBC` and `BaselineBC*`. We use `MineLMBC` [22] for enumerating bicliques from a static graph. Table 2 shows a comparison of the runtimes of `DynamicBC` with `BaselineBC` and `BaselineBC*`. From the table, it is clear that `DynamicBC` is orders of magnitude faster than `BaselineBC` and many times faster than `BaselineBC*`. For instance, for adding 625 batches of edges starting from `lastfm-song-init`, `DynamicBC` takes about 93 sec., `BaselineBC` about $7,920$ sec., and `BaselineBC*` about $1,740$ sec.



(a) `epinions-rating-init` (b) `lastfm-song-init`

(c) `movielens-10M-init` (d) `wiktionary-init`

Fig. 6: **Computation time (in sec.) for total change vs. size of total change. The left y-axis shows the change and the right y-axis shows the computation time.**

TABLE 1: **Summary of the input graphs.**

Dataset	Nodes	Edges (start)	Edges (stop)	Edges (original graph)
`epinions-rating-init`	876,252	0	1,210,000	13,668,320
`lastfm-song-init`	1,085,612	0	361,500	4,413,834
`movielens-10M-init`	80,555	0	8,500	10,000,054
`wiktionary-init`	2,123,868	0	235,000	5,573,038

TABLE 2: **For each algorithm, the number shows the cumulative computation time for the number (in the parenthesis) of batch additions incrementally.**

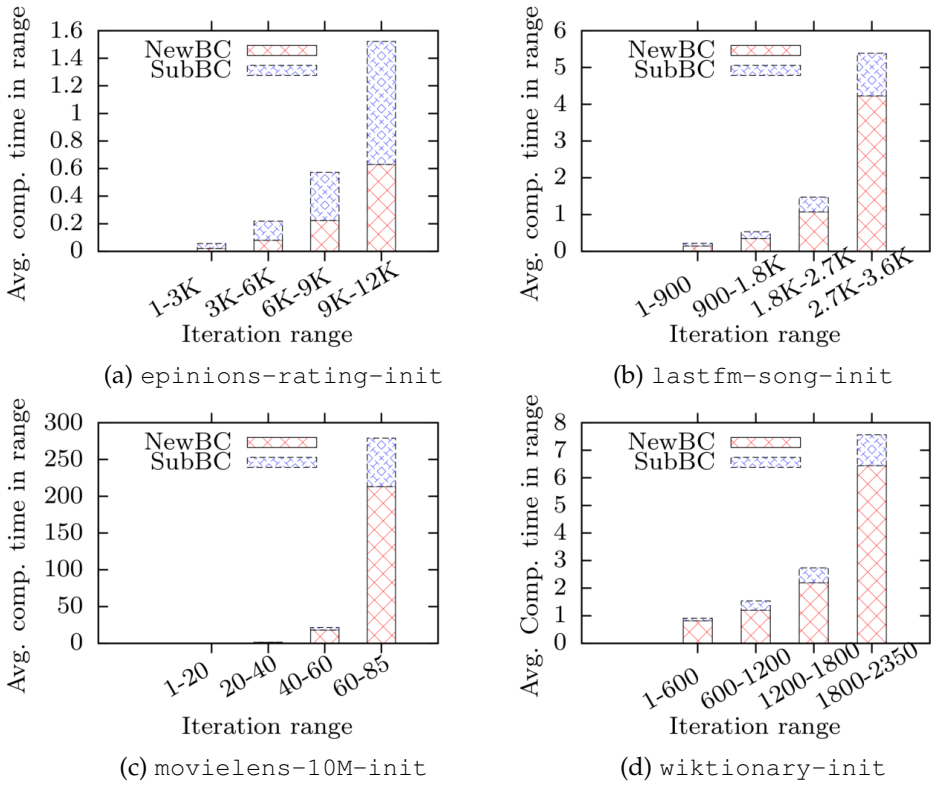
Initial-graph	`DynamicBC`	`BaselineBC`	`BaselineBC`*
`epinions-rating-init` (424)	2 sec.	7,200 sec.	17 sec.
`lastfm-song-init` (625)	93 sec.	7,920 sec.	1,740 sec.
`movielens-10M-init` (58)	7 min.	out of memory after 23 min.	10.8 min.
`wiktionary-init` (494)	8 min.	out of memory after 96 min.	149.15 min.

TABLE 3: **Total computation time in hours for different batch sizes. The total time is split into two numbers. The first number is the time for new maximal bicliques and the second number is the time for subsumed maximal bicliques.**

Initial-graph	**batch-size**-1	**batch-size**-10	**batch-size**-100
`epinions-rating-init`	1 (0.9 + 0.1)	1 (0.8 + 0.2)	2 (0.8 + 1.2)
`lastfm-song-init`	1.5 (1.45 + 0.09)	1.8 (1.5 + 0.3)	1.9 (1.5 + 0.4)
`movielens-10M-init`	0.4 (0.36+0.04)	0.6 (0.5 + 0.1)	2.1 (1.6 + 0.5)
`wiktionary-init`	1.8 (1.7 + 0.1)	1.8 (1.7 + 0.1)	2 (1.7 + 0.3)

TABLE 4: **Total computation time in hour by varying the threshold size s.**

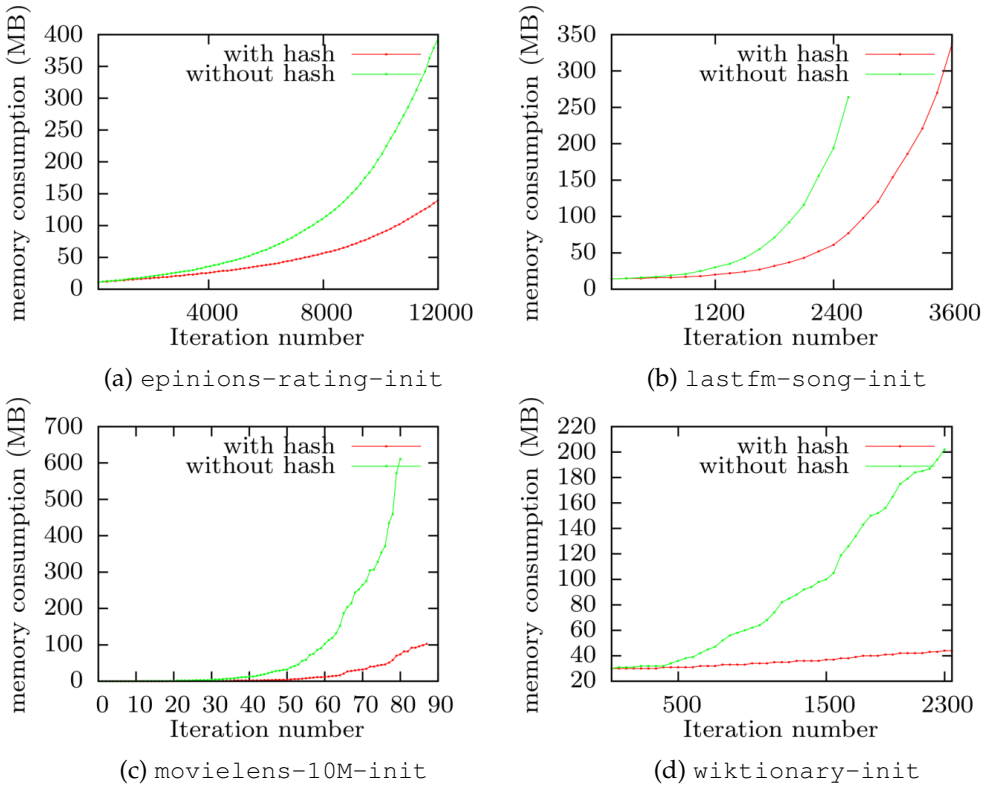
Initial-graph	$s = 2$	$s = 4$	$s = 6$	$s = 8$	$s = 10$	$s = 12$
`epinions-rating-init`	1.2	0.9	0.7	0.4	0.3	0.3
`lastfm-song-init`	1.6	1.3	0.7	0.4	0.3	0.3
`movielens-10M-init`	2.1	1.9	1.6	0.9	0.3	0.1
`wiktionary-init`	1.5	1	0.7	0.5	0.4	0.4



(a) `epinions-rating-init` (b) `lastfm-song-init`

(c) `movielens-10M-init` (d) `wiktionary-init`

Fig. 7: **Computation time (in sec.) broken down into time for new and subsumed bicliques.**



(a) `epinions-rating-init` (b) `lastfm-song-init`

(c) `movielens-10M-init` (d) `wiktionary-init`

Fig. 8: **Memory consumption (in MB) with and without using hash function.**

Computation Time per Batch of Edges: Fig. 6 shows the computation time (per batch) versus iteration number where one batch is added in each iteration. From the plots, we observe an increasing trend in computation time with the iteration number. There are two reasons for this. One is that with more iterations, the graph becomes denser, and

the average degree increases. This contributes to the runtime of computing new maximal bicliques, as is predicted by theory (Theorem 3). Another reason is that the size of change in the maximal bicliques typically increases as more edges are added to the graph, as can be seen from the figure. Whenever the size of change in maximal bicliques drops, the computation time also drops. Fig. 7 shows the breakdown of computation time of `DynamicBC` into time taken for enumerating new bicliques (`NewBC`) and time taken for enumerating subsumed bicliques (`SubBC`). Observe that the average computation time (where the average is taken over a range of iterations) increases for both new maximal bicliques and subsumed bicliques as more batches are added, for the same reasons as above.

Change-Sensitiveness: Fig. 6 shows the computation time and the size of change, as measured in terms of the number of nodes as well as number of edges. We observe that the computation time increases as the size of change increases and decreases as the size of change decreases. But the relationship is not exactly linear. This is because the computation time depends also on the degree of vertices of the graph, which increases as more edges are inserted.

Memory Consumption: Fig. 8 shows the memory consumption of `DynamicBC`. Since `SubBC` needs to maintain the maximal bicliques in memory for computing subsumed bicliques, we report the memory consumption in two cases: (1) when the maximal bicliques are stored in memory, (2) when hash signatures of maximal bicliques are stored in memory. Storing signatures consumes less memory than storing actual bicliques (by storing the node sets) as the signatures have fixed size (64 bits) no matter the size of the bicliques. This difference in memory is also clear in the plots. The difference in memory consumption is not prominent during the initial iterations because the sizes of maximal bicliques are much smaller during initial iterations.

Effect of Batch Size on Cumulative Computation Time: Table 3 shows the cumulative computation time for different graphs when we use different batch sizes. We observe that overall the total computation time increases when the batch size increases. The reason is the computation time for subsumed bicliques, which increases with increasing batch size, while the computation time for the new maximal bicliques remains almost the same across different batch sizes except `movielens-10M-init`. In case of `movielens-10M-init`, we observed that the time for computing new maximal bicliques also increased with the batch size. The reason is that the algorithm for new maximal bicliques can enumerate the same (new) maximal biclique multiple times, for considering new edges. Such duplicates are suppressed before emitting, but contribute to additional runtime. For this graph, the number of duplicates increased considerably for a batch size of say, 100.

The time complexity for `SubBC` has (in the worst case) an exponential dependence on the batch size. Therefore, the computation time for subsumed bicliques tends to increase with an increase in the batch size. However, with a very small batch size (such as 1 or 10), the cost of enumerating subsumed bicliques was mostly dominated by the cost of enumerating new maximal bicliques.

Effect of Size Threshold on Computation Time: We also consider maintaining maximal bicliques with specified size threshold s, where it is required that each bipartition has size at least s. Table 4 shows the cumulative computation time by varying the threshold s. As expected, the cumulative computation time decreases as the size threshold s increases, since there is more pruning possible during the depth-first-search performed by Algorithm `MineLMBC`.

6 Conclusion

In this work, we presented a change-sensitive algorithm for enumerating changes in the set of maximal bicliques in dynamic graph. The performance of this algorithm is proportional to the magnitude of change in the set of maximal bicliques – when the change is small, the algorithm runs faster, and when the change is large, it takes a proportionally longer time. We present near-tight bounds on the maximum possible change in the set of maximal bicliques, due to a change in the set of edges in the graph. Our experimental evaluation shows that the algorithm is efficient in practice, and scales to graphs with millions of edges. This work leads to natural open questions (1) Can we design more efficient algorithms for enumerating the change, especially for enumerating subsumed bicliques? (2) Can we parallelize the algorithm for enumerating the change in maximal bicliques?

References

[1] Epinions product ratings network dataset – KONECT. http://konect.uni-koblenz.de/networks/epinions-rating, Apr. 2017.

[2] Last.fm song network dataset – KONECT. http://konect.uni-koblenz.de/networks/lastfm_song, Apr. 2017.

[3] Movielens 10m network dataset – KONECT. http://konect.uni-koblenz.de/networks/movielens-10m_rating, Apr. 2017.

[4] Wiktionary (en) network dataset – KONECT. http://konect.uni-koblenz.de/networks/edit-enwiktionary, Apr. 2017.

[5] 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.

[6] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB Journal*, 23(2):175–199, 2014.

[7] P. Damaschke. Enumerating maximal bicliques in bipartite graphs with favorable degree sequences. *Information Processing Letters*, 114(6):317–321, 2014.

[8] A. Das, M. Svendsen, and S. Tirthapura. Change-sensitive algorithms for maintaining maximal cliques in a dynamic graph. *CoRR*, abs/1601.06311, 2016.

[9] V. M. Dias, C. M. De Figueiredo, and J. L. Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1):240–248, 2005.

[10] 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.

[11] A. C. Driskell, C. An, J. G. Burleigh, M. M. McMahon, B. C. O'Meara, and M. J. Sanderson. Prospects for building the tree of life from large sequence databases. *Science*, 306(5699):1172–1174, 2004.

[12] D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994.

[13] A. Gély, L. Nourine, and B. Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete applied mathematics*, 157(7):1447–1459, 2009.

[14] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005.

[15] R. A. Hanneman and M. Riddle. Introduction to social network methods, 2005.

[16] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[17] A. Java, X. Song, T. Finin, and B. L. Tseng. Why we twitter: An analysis of a microblogging community. In *WebKDD/SNA-KDD*, pages 118–138, 2007.

[18] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer networks*, 31(11):1481–1493, 1999.

[19] S. Lehmann, M. Schwartz, and L. K. Hansen. Biclique communities. *Physical Review E*, 78(1):016108, 2008.

[20] J. Li, H. Li, D. Soh, and L. Wong. A correspondence between maximal complete bipartite subgraphs and closed patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 146–156. Springer, 2005.

[21] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *TKDE*, 26(10):2453–2465, 2014.

[22] G. Liu, K. Sim, and J. Li. Efficient mining of large maximal bicliques. In *Data warehousing and knowledge discovery*, pages 437–448. Springer, 2006.

[23] D. Lo, D. Surian, K. Zhang, and E.-P. Lim. Mining direct antagonistic communities in explicit trust networks. In *CIKM*, pages 1013–1018, 2011.

[24] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *SWAT*, pages 260–272. 2004.

[25] O. Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[26] A. P. Mukherjee and S. Tirthapura. Enumerating maximal bicliques from a large graph using mapreduce. In *IEEE BigData Congress*, pages 707–716, 2014.

[27] A. P. Mukherjee and S. Tirthapura. Enumerating maximal bicliques from a large graph using mapreduce. *IEEE Transactions on Services Computing*, 10(5):771–784, 2017.

[28] A. P. Mukherjee, P. Xu, and S. Tirthapura. Enumeration of maximal cliques from an uncertain graph. *IEEE transactions on knowledge and data engineering*, 29(3):543–555, 2017.

[29] T. Murata. Discovery of user communities from web audience measurement data. In *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on*, pages 673–676. IEEE, 2004.

[30] N. Nagarajan and C. Kingsford. Uncovering genomic reassortments among influenza strains by enumerating maximal bicliques. In *Bioinformatics and Biomedicine, 2008. BIBM'08. IEEE International Conference on*, pages 223–230. IEEE, 2008.

[31] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46, 1999.

[32] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.

[33] E. Prisner. Bicliques in graphs i: Bounds on their number. *Combinatorica*, 20(1):109–117, 2000.

[34] J. E. Rome and R. M. Haralick. Towards a formal concept analysis approach to exploring communities on the world wide web. In *Formal Concept Analysis*, volume 3403 of *LNCS*, pages 33–48. 2005.

[35] M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley. Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Mol. Biol. Evol.*, 20(7):1036–1042, 2003.

[36] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013.

[37] N. Simsiri, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Work-efficient parallel union-find with applications to incremental graph connectivity. In *European Conference on Parallel Processing*, pages 561–573. Springer, 2016.

[38] M. Svendsen, A. P. Mukherjee, and S. Tirthapura. Mining maximal cliques from a large graph using mapreduce: Tackling highly uneven subproblem sizes. *J. Parallel Distrib. Comput.*, 79-80:104–114, 2015.

[39] M. Thorup. Decremental dynamic connectivity. *Journal of Algorithms*, 33(2):229–243, 1999.

[40] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[41] P. Valtchev, R. Missaoui, and R. Godin. A framework for incremental generation of closed itemsets. *Discrete Applied Mathematics*, 156(6):924–949, 2008.

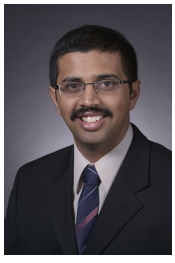
[42] C. Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1757–1769. SIAM, 2013.

[43] Y. Xu, J. Cheng, A. W.-C. Fu, and Y. Bu. Distributed maximal clique computation. In *IEEE BigData Congress*, pages 160–167, 2014.

[44] C. Yan, J. G. Burleigh, and O. Eulenstein. Identifying optimal incomplete phylogenetic data sets from sequence databases. *Mol. Phylogenet. Evol.*, 35(3):528–535, 2005.

[45] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics*, 15(1):1, 2014.

Apurba Das is a Ph.D. student in Computer Engineering at Iowa State University. He received his Masters in Computer Science from Indian Statistical Institute, Kolkata and has worked as a software developer at Ixia. His research interests are in the area of graph mining, dynamic and streaming graph algorithms, and large scale data analysis.



Dr. Srikanta Tirthapura received his Ph.D. in Computer Science from Brown University in 2002, and his B.Tech. in Computer Science and Engineering from IIT Madras in 1996. He is the Kingland Professor of Data Analytics in the department of Electrical and Computer Engineering at Iowa State University. He has worked at Oracle Corporation, and is a recipient of the IBM Faculty Award, and the Warren Boast Award for excellence in Undergraduate Teaching. His research interests include algorithms for large-scale data analysis, stream computing, and cybersecurity.