Incremental Maintenance of Maximal Cliques in a Dynamic Graph

Apurba Das · Michael Svendsen · Srikanta Tirthapura

Abstract We consider the maintenance of the set of all maximal cliques in a dynamic graph that is changing through the addition or deletion of edges. We present nearly tight bounds on the magnitude of change in the set of maximal cliques, as well as the first change-sensitive algorithms for clique maintenance, whose runtime is proportional to the magnitude of the change in the set of maximal cliques. We present experimental results showing these algorithms are efficient in practice, and are faster than prior work by two to three orders of magnitude.

1 Introduction

Graphs are widely used in modeling linked data, and there has been tremendous interest in efficient methods for finding patterns in graphs, an area often called "graph mining". A fundamental task in graph mining is the identification of *dense subgraphs*, which are groups of vertices that are tightly interconnected.

Many applications need to identify dense subgraphs from an evolving graph that is changing with time as new edges are added and old edges are deleted. Examples include real-time identification of stories from Twitter [1] through mining dense subgraphs from an evolving graph on entities, and the maintenance of common intervals among genomes [5] through mining maximal cliques in an ap-

Apurba Das
Iowa State University
E-mail: adas@iastate.edu

Michael Svendsen
Iowa State University
E-mail: michael.sven5@gmail.com

Srikanta Tirthapura
Iowa State University
E-mail: snt@iastate.edu

propriately defined dynamic graph. More broadly, identifying dense structures in a graph is applicable to any task that needs to identify and analyze communities with a network, such as the analysis of communities among users in microblogging platforms [18], identification of groups of closely linked people in a social network [15, 23, 26], identification of web communities [14, 22, 33], and even in the construction of the Phylogenetic Tree of Life [9, 34, 44].

Most current methods for identifying dense subgraphs are designed for a static graph. Suppose we used a method designed for a static graph to handle a dynamic graph. If the input graph changes slightly, say, by the addition of a few edges, it is necessary to enumerate all dense subgraphs all over again, even though the set of dense subgraphs may have only changed slightly due to the addition of the new edges. This repeated and redundant work is a source of serious inefficiency, so that methods designed for static graphs are not applicable to a graph that is changing frequently. Different methods are needed, which can handle changes to a graph more efficiently. From a foundational perspective, identifying dense structures in a graph has been a problem of longstanding interest in computer science, but even basic questions remain unanswered on dynamic graphs.

We consider the maintenance of the set of *maximal cliques* in a dynamic graph. The maximal clique is perhaps the most fundamental and widely studied dense subgraph. Let $G = (V, E)$ be an undirected unweighted graph on vertex set V and edge set E. A clique in G is a set of vertices $C \subseteq V$ such that any two vertices in C are connected to each other in G. A clique is called maximal if it is not a proper subset of any other clique. Let $\mathcal{C}(G)$ denote the set of maximal cliques in G. Many applications benefit from efficient maintenance of maximal cliques in a dynamic graph, such as described in the work of Chateau et al. [5] on maintaining common intervals among genomes, Duan et al. [10] on

incremental k-clique clustering, Hussain et al. [17] on maintaining the maximum range-sum query over a point stream.

Suppose that we started from a graph $G = (V, E)$ and the state of the graph changed to $G' = (V, E \cup H)$ through an addition of a set of new edges H to the set of edges in the graph G. See Figure 1 for an example. Let $\Lambda^{new}(G, G') = \mathcal{C}(G') \setminus \mathcal{C}(G)$ denote the set of maximal cliques that were newly formed in going from G to G', and $\Lambda^{del}(G, G') = \mathcal{C}(G) \setminus \mathcal{C}(G')$ denote the set of cliques that were maximal in G but are no longer maximal in G'. Let $\Lambda(G, G') = \Lambda^{new}(G, G') \cup \Lambda^{del}(G, G')$ denote the symmetric difference of $\mathcal{C}(G)$ and $\mathcal{C}(G')$. We ask the following questions:

– How large can the size of $\Lambda(G, G')$ be? To systematically study the problem of maintaining maximal cliques in a dynamic graph, we first need to understand the magnitude of change in the set of maximal cliques.

– What are efficient methods to compute $\Lambda(G, G')$? Is it possible to have methods that compute $\Lambda(G, G')$ quickly in cases when the size of $\Lambda(G, G')$ is small, and take longer when it is large? Do these methods scale to large graphs?

1.1 Contributions

(A) Magnitude of Change in the Set of Maximal Cliques: We present a tight analysis of the magnitude of change in the set of maximal cliques in a graph, when a set of edges are added. When a set of edges H is added to graph $G = (V, E)$ resulting in graph $G' = G \cup H = (V, E \cup H)$.

(A.1) We present nearly matching upper and lower bounds on the maximum size of $\Lambda(G, G \cup H)$, taken across all possible graphs G and edge sets H. Let $f(n)$ denote the maximum number of maximal cliques in a graph on n vertices. A result of Moon and Moser [29] shows that $f(n)$ is approximately $3^{n/3}$. We show that by the addition of a small number of edges to the graph G on n vertices, it is possible to cause a change of nearly $2f(n) \approx 2 \cdot 3^{n/3}$. We also note that this is an upper bound on the magnitude of $\Lambda(G, G')$. We present this analysis in Theorem 3.

(A.2) We encountered an error in the 50-year old result of Moon and Moser [29] on the number of maximal cliques in a graph, which is directly relevant to our bounds on the change in the set of maximal cliques. We present our correction to their result in Observation 1.

It is easy to see that the set of maximal cliques can change by very little upon the addition of edges. For instance, adding a single edge between two vertices that are part of different components can lead to only a single

new maximal clique being added (the clique consisting of a single edge), and no maximal cliques subsumed, so that the total change in the set of maximal cliques is 1. Thus, we note that the magnitude of the change can vary significantly from one input instance to another.

(B) Algorithm for Maintaining Maximal Cliques: We present incremental and decremental algorithms for maintaining the set of maximal cliques of a dynamic graph. We describe result on incremental algorithms here. The results for decremental algorithms are similar.

(B.1) We present algorithms that take as input G and H, and enumerate the elements of $\Lambda(G, G')$ in time proportional to the size of $\Lambda(G, G')$, i.e. the magnitude of the change in the set of maximal cliques. We refer to such algorithms as *change-sensitive* algorithms. To our knowledge, these are the first provably change-sensitive algorithms for maintaining the set of maximal cliques in a dynamic graph. The time taken for enumerating newly formed cliques $\Lambda^{new}(G, G')$ is $O(\Delta^3 \rho |\Lambda^{new}(G, G')|)$ where Δ is the maximum degree of a vertex in G' and ρ is the number of edges in H. The time taken for enumerating subsumed cliques $\Lambda^{del}(G, G')$ is $O(2^\rho |\Lambda^{new}(G, G')|)$. Note that when ρ, the size of a batch of edges, is logarithmic in Δ, the cost of enumerating subsumed cliques is of the same order as that of enumerating new cliques.

Our algorithm is based on a careful exploration of a subgraph of G that is local to the set of edges that have been added. Importantly, it does not iterate through existing maximal cliques in the graph. Instead, it directly outputs the maximal cliques that have changed (either added or subsumed). Based on theoretically-efficient algorithms, we present a practical algorithm `IMCE` for enumerating new and subsumed cliques, and an efficient implementation.

(B.2) Our methods extend to the decremental case, to handle deletion of edges from the graph. They can also be applied to the fully dynamic case, where the change includes both the addition and deletion of edges from the graph. However, the fully dynamic case is not provably change-sensitive, as discussed in Section 4.4.

(C) Experimental Evaluation We present empirical evaluation of our algorithm using real world dynamic graphs as well as synthetic graphs. Our experimental study shows that `IMCE` can enumerate change in maximal cliques in a large graph with of the order of a hundred thousand vertices and millions of edges within a few seconds. Our comparison with prior and recent works show that `IMCE` significantly outperform prior solutions, including ones due to Stix [37], Ottosen and Vomlel [32], and Sun et al. [38]. For example, on the `flickr-growth` graph, our algorithms are faster
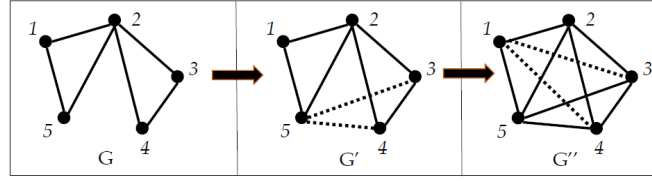
Fig. 1: Change in maximal cliques due to addition of edges. On the left is the initial graph G with maximal cliques $\{1, 2, 5\}$ and $\{2, 3, 4\}$; On the middle is the graph G' after adding edges $(3, 5)$ and $(4, 5)$ to G resulting in new maximal clique $\{2, 3, 4, 5\}$ and only subsumed maximal clique $\{2, 3, 4\}$; On the right is the graph G'' after adding edges $(1, 3)$ and $(1, 4)$ to G' resulting in new maximal clique $\{1, 2, 3, 4, 5\}$ and subsumed cliques $\{1, 2, 5\}$ and $\{2, 3, 4, 5\}$.

than [32, 37, 38] by a factor of more than a thousand. On the `flickr-growth` graph, in order to maintain the set of maximal cliques over the insertion of 250 batches of 100 edges each, IMCE took about 40 ms, while prior techniques took anywhere from 5 mins to 2 hrs. Further details are in Section 5.

1.2 Prior and Related Work

Maximal Clique Enumeration in a Static Graph. There is substantial prior work on enumerating maximal cliques in a static graph, starting from the algorithm based on depth-first-search due to Bron and Kerbosch [4]. A significant improvement to [4] is presented in Tomita et al. [41], leading to worst-case optimal time complexity $O(3^{n/3})$ for an n vertex graph [29]. Other work on refinements of [4, 41] include [20], who presents several strategies for pivot selection to enhance the algorithm in [4], and a fixed parameter tractable algorithm parameterized by the graph degeneracy [11, 12].

There is a class of algorithms for enumerating structures (such as maximal cliques) in a static graph whose time complexity is proportional to the size of the output – such algorithms are called "output-sensitive" algorithms. Many output-sensitive structure enumeration algorithms for static graphs, including [7, 27, 42], can be seen as instances of a general technique called "reverse search" [2]. The current best bound on the time complexity of output-sensitive maximal clique enumeration on a dense graph $G = (V, E)$ is due to [27] which runs with $O(M(n))$ time delay (the interval between outputting two maximal cliques), where $M(n)$ is the time complexity for multiplying two $n \times n$ matrix, which is $O(n^{2.376})$. Further work in this direction includes [21] and [19], who consider the enumeration of maximal independent sets in lexicographic order, [6], who consider the external memory model, and [31], who consider uncertain graphs. Extensions to parallel frameworks such as MapReduce or MPI are presented in [30, 39].

Maximal Clique Enumeration in a Dynamic Graph. In [37], the authors present algorithms for tracking new and

subsumed maximal cliques in a dynamic graph when a single edge is added to the graph. These algorithms are not proved to be change-sensitive, even for a single edge. The algorithm due to Stix [37] for enumerating new maximal cliques needs to consider (and filter out) maximal cliques in the original graph that remain unaffected due to addition of new edge. This can be wasteful, in terms of update time. Hence, such an algorithm cannot be change-sensitive. For example, consider the case of a graph growing from an empty graph on 10 vertices to a clique on 10 vertices. Only one new maximal clique has been formed by this batch, but numerous maximal cliques arise during intermediate steps – if all these are enumerated, then the time complexity of enumeration is inherently large, even though the magnitude of change is small.

Ottosen and Vomlel [32] present an algorithm to enumerate the change in set of maximal cliques, based on running a maximal clique enumeration algorithm on a smaller graph. Their algorithm supports addition of a set of edges all at once. In contrast with our work, there are no provable performance bounds for this algorithm. Another difference is that the algorithm of [32] may not maintain the exact change in the set of maximal cliques, in certain cases, while our algorithms can maintain the change in the set of maximal cliques exactly.

Sun et al. [38] present an algorithm for enumerating the change in set of maximal cliques, based on iterating over the set of maximal cliques of the original graph to derive the set of maximal cliques of the updated graph. This need to iterate over currently existing cliques makes the algorithm expensive, especially for cases when the set of maximal cliques does not change significantly due to the update in edge set.

Prior algorithms for maximal clique enumeration on a dynamic graph are not proved to be change-sensitive, and do not provide a provable bound on the cost to enumerate the change, or on the magnitude of the change.

Other Queries on a Dynamic Graph. Other works on maintaining dense structures on a dynamic graph include methods for the maintenance of k-cores [25, 35], k-truss communities [16], densest subgraph [3, 28], and maximal bicliques in a bipartite graph [8]. Bicliques are complete

structures in bipartite graphs, and the theory on the number of substructures and enumeration algorithms is substantially different. The other structures: k-core, k-truss, and densest subgraph, are different from maximal cliques in that they do not require complete connectivity among different vertices within the structure.

Roadmap: We present preliminaries in Section 2, followed by bounds on magnitude of change in Section 3, algorithms for enumerating the change in Section 4, and experimental results in Section 5.

2 Preliminaries

We consider a simple undirected graph without self loops or multiple edges. For graph G, let $V(G)$ denote the set of vertices in G and $E(G)$ denote the set of edges in G. Let n denote the size of $V(G)$, and m denote the size of $E(G)$. For vertex $u \in V(G)$, let $\Gamma_G(u)$ denote the set of vertices adjacent to u in G. When the graph G is clear from the context, we use $\Gamma(u)$ to mean $\Gamma_G(u)$. For edge $e = (u, v) \in E(G)$, let $G - e$ denote the graph obtained by deleting e from $E(G)$, but retaining vertices u and v in $V(G)$. Similarly, let $G + e$ denote the graph obtained by adding edge e to $E(G)$. For edge set H, let $G + H$ ($G - H$) denote the graph obtained by adding (subtracting) all edges in H to (from) $E(G)$. Let $\Delta(G)$ denote the maximum degree of a vertex in G. When the context is clear, we use Δ to mean $\Delta(G)$. For vertex $v \in V(G)$, let $G - v$ denote the induced subgraph of G on the vertex set $V(G) - \{v\}$, i.e. the graph obtained from G by deleting v and all its incident edges. Let $\mathcal{C}_v(G)$ denote the set of maximal cliques in G containing v.

Change-Sensitive Algorithms: 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. Note that the notion of "change-sensitive" for a dynamic graph algorithm is similar to the notion of "output-sensitive" in the static graph algorithm where the time complexity is proportional to the size of output times polynomial in other parameters like degree, number of edges etc. For example, see Theorem 2.

An algorithm for a dynamic graph is called *incremental* if it can efficiently handle insertion of edges, *decremental* if it can handle deletion of edges, and *fully dynamic* if it can handle both insertions and deletions. For example, a parallel algorithm due to Simsiri et al. [36] is an incremental algorithm for graph connectivity, an algorithm due to Thorup [40] is a decremental algorithm, and one due to Wulff-Nilsen [43] is a fully dynamic algorithm. Our algorithms can be viewed as a change-sensitive incremental algorithm for maximal cliques, and a change-sensitive decremental algorithm for maximal cliques.

Results for Static Graphs: We present some known results about maximal cliques on static graph. Nearly 50 years ago, Moon and Moser [29] considered the question: "what is the maximum number of maximal cliques that can be present in an undirected graph on n vertices", and gave the following answer. Let $f(n)$ denote the maximum possible number of maximal cliques in a graph on n vertices. A graph on n vertices that achieves $f(n)$ maximal cliques is called a "Moon-Moser" graph.

Theorem 1 (Theorem 1, Moon and Moser, [29])

$$
\begin{aligned}
f(n) &= 3^{\frac{n}{3}} && \text{if } n \mod 3 = 0 \\
&= 4 \cdot 3^{\frac{n-4}{3}} && \text{if } n \mod 3 = 1 \\
&= 2 \cdot 3^{\frac{n-2}{3}} && \text{if } n \mod 3 = 2
\end{aligned}
$$

We use as a subroutine an output-sensitive algorithm for enumerating all maximal cliques within a (static) graph, using time proportional to the number of maximal cliques. There are multiple such algorithms, for example, due to Tsukiyama et al. [42], and due to Makino and Uno [27]. We use the following result due to Chiba and Nishizeki since it provides one of the best possible time complexity bounds for general graphs. Better results are possible for dense graphs [27] and our algorithm can use other methods as a subroutine also.

Theorem 2 (Chiba and Nishizeki, [7]) *There is an algorithm* $\mathtt{MCE}(G)$ *that enumerates all maximal cliques in graph* G *using time* $O(\alpha m \mu)$ *where* μ *is the number of maximal cliques in* G *and* α *is the arboricity of* G[1] *The total space complexity of the algorithm is* $O(n + m)$.

3 Magnitude of Change

From prior work [29], the maximum number of maximal cliques in an n-vertex graph, denoted by $f(n)$ is known (see Theorem 1). The result of [29] is relevant for static graphs. In the case of a dynamic graph, a different question is more relevant: *what is the maximum change in the set of maximal cliques, that can result from the addition of edges to the graph?* This will give us a bound on the worst case complexity of enumerating the change in the set of maximal cliques.

3.1 Maximum Possible Change in Maximal Cliques

We consider the maximum change in the set of maximal cliques upon the addition of edges to the graph. For an integer n, let $\lambda(n)$ be the maximum size of $\Lambda(G, G + H)$ taken

---

[1] The arboricity of a graph is no more than the maximum vertex degree of the graph, but could be significantly lesser.

over all possible n vertex graphs G and edge sets H. We present the following result with nearly tight bounds on the value of $\lambda(n)$. Interestingly, our results show that it is possible to change the set of maximal cliques by as much as $\approx 2 \cdot 3^{n/3}$ by the addition of only a few edges to the graph.

Theorem 3

$$\frac{16}{9} f(n) \leq \lambda(n) < 2f(n) \qquad if \ (n \mod 3) = 0$$

$$\lambda(n) = 2f(n) \qquad if \ (n \mod 3) = 1$$

$$\frac{11}{6} f(n) \leq \lambda(n) < 2f(n) \qquad if \ (n \mod 3) = 2$$

Proof We first note that $\lambda(n) \leq 2f(n)$ for any integer n. To see this, note that for any graph G on n vertices and edge set H, it must be true from Theorem 1 that $|\mathcal{C}(G)| \leq f(n)$ and $|\mathcal{C}(G+H)| \leq f(n)$. Since $|\Lambda^{new}(G, G+H)| \leq |\mathcal{C}(G+H)|$ and $|\Lambda^{del}(G, G+H)| \leq |\mathcal{C}(G)|$, we have $|\Lambda(G, G+H)| = |\Lambda^{new}(G, G+H)| + |\Lambda^{del}(G, G+H)| \leq |\mathcal{C}(G)| + |\mathcal{C}(G+H)| \leq 2f(n)$.

The result of Moon and Moser [29] states that for $n \geq 2$, there is only one graph H_n on n vertices (subject to isomorphism) that has $f(n)$ maximal cliques. We show below that there is an error in this result for the case $(n \mod 3) = 1$. Note that the result is still true in the cases $(n \mod 3)$ equals 0 or 2. Thus for the cases where $(n \mod 3)$ is 0 or 2, adding or deleting edges from H_n leads to a graph with fewer than $f(n)$ maximal cliques, so that we can never achieve a change of $2f(n)$ maximal cliques. Thus we have that for $(n \mod 3)$ equal to 0 or 2, $\lambda(n)$ is strictly less than $2f(n)$. The case of $(n \mod 3) = 1$ is discussed separately (see Observation 1 below).

We next show that there exists a graph G on n vertices and an edge set H such that the size of $\Lambda(G, G+H)$ is large. See Figure 2 for an example. Graph G is constructed on n vertices as follows. Let $\varepsilon > 3$ be an integer. Choose ε vertices in $V(G)$ into set V_1. Let $V_2 = V \setminus V_1$. Edges of G are constructed as follows.

- Each vertex in V_1 is connected to each vertex in V_2.
- Edges are added among vertices of V_2 to make the induced subgraph on V_2 a Moon-Moser graph on $(n - \varepsilon)$ vertices. Let G_2 denote this induced subgraph on V_2, which has $f(n - \varepsilon)$ maximal cliques.
- There are no edges among vertices of V_1 in G.

It is clear that for each maximal clique c in G_2 and vertex $v \in V_1$, there is a maximal clique in G by adding v to c. Thus the number of maximal cliques in G is $|V_1| \cdot |\mathcal{C}(G_2)|$. Hence we have

$$|\mathcal{C}(G)| = \varepsilon \cdot f(n - \varepsilon) \qquad (1)$$

To graph G, we add the edge set H, constructed as follows. H consists of edges connecting vertices in V_1, to form

a Moon-Moser graph on ε vertices. Let $G' = G + H$. We note that $\mathcal{C}(G)$ and $\mathcal{C}(G')$ are disjoint sets. To see this, note that each maximal clique in G contains exactly one vertex from V_1, since no two vertices in V_1 are connected to each other in G. On the other hand, each maximal clique in G' contains more than one vertex from V_1, since each vertex $v \in V_1$ is connected to at least one other vertex in V_1 in G'. Hence, $\Lambda(G, G') = \mathcal{C}(G) \cup \mathcal{C}(G')$, and

$$|\Lambda(G, G')| = |\mathcal{C}(G)| + |\mathcal{C}(G')| \qquad (2)$$

To compute $|\mathcal{C}(G')|$, note that since each vertex in V_1 is connected to each vertex in V_2, for each maximal clique in $G'(V_1)$ and each maximal clique in $G'(V_2)$, we have a unique maximal clique in G'. There are $f(\varepsilon)$ maximal cliques in $G'(V_1)$ and $f(n - \varepsilon)$ maximal cliques in $G'(V_2)$, and hence we have

$$|\mathcal{C}(G')| = f(\varepsilon) \cdot f(n - \varepsilon) \qquad (3)$$

Putting together Equations 1, 2, and 3 we get

$$|\Lambda(G, G')| = (\varepsilon + f(\varepsilon)) \cdot f(n - \varepsilon) \qquad (4)$$

Let $F(\varepsilon) = (\varepsilon + f(\varepsilon))f(n - \varepsilon)$. We compute the value of $\varepsilon(> 3)$ at which $F(\varepsilon)$ is maximized. To do this, we consider three different cases depending on the value of $(n \mod 3)$, and omit the calculations. If $n \mod 3 = 0$, $F(\varepsilon)$ is maximized at $\varepsilon = 4$ and the maximum value $F(4) = \frac{16}{9}f(n)$. If $n \mod 3 = 1$, $F(\varepsilon)$ is maximized at $\varepsilon = 4$ and $F(4) = 2f(n)$. And finally if $n \mod 3 = 2$, $F(\varepsilon)$ is maximized at $\varepsilon = 5$ and $F(5) = \frac{11}{6}f(n)$. This completes the proof. $\square$

3.2 An Error in a Result of Moon and Moser (1965)

Moon and Moser [29], in Theorem 2 in their paper, claim "For any $n \geq 2$, if a graph G has n nodes and $f(n)$ cliques, then G must be equal to H_n", where H_n is a specific graph, described below. We found that this theorem is incorrect for the case when $(n \mod 3) = 1$.

The error is as follows (see Figure 3). For $(n \mod 3) = 1$, the graph H_n is constructed on vertex set $V_n = \{1, 2, \ldots, n\}$ by taking vertices $\{1, 2, 3, 4\}$ into a set S_0 and dividing the remaining vertices into groups of three, as sets $S_1, S_2, \ldots, S_{\frac{n-4}{3}}$. In graph H_n, edges are added between any two vertices u, v such that $u \in S_i$, $v \in S_j$ and $i \neq j$. This graph H_n has $4 \cdot 3^{\frac{n-4}{3}}$ maximal cliques, since we can make a maximal clique by choosing a vertex from S_0 (4 ways), and one vertex from each $S_i, i > 0$ (3 ways for each such $S_i, i = 1 \ldots (n-4)/3$).

Contradicting Theorem 2 in [29], we show there is another graph G_n that is different from H_n, but still has the same number of maximal cliques. G_n is the same as H_n,
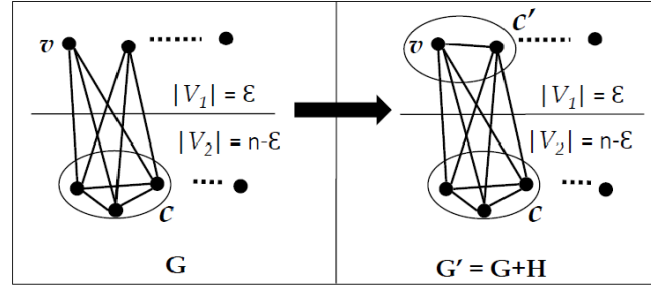
Fig. 2: Construction showing a large change in set of maximal cliques when a few edges are added. V_1 is the set of vertices above the horizontal line and V_2 is the set of vertices below the horizontal line where $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \phi$. On the left is G, the original graph with n vertices where each vertex in V_1 is connected to each vertex in V_2, and V_1 is an independent set. In G, the induced subgraph G_2 on vertex set V_2 forms a Moon–Moser graph. On the right is G', the graph formed after adding edge set H to G such that the induced subgraph on vertex set V_1 becomes a Moon–Moser graph. Let c be a clique in G_2, and c' a new clique in G' formed among vertices in V_1. Note that $c \cup \{v\}$ was a maximal clique in G, and is now subsumed by a new maximal clique $c \cup c'$.
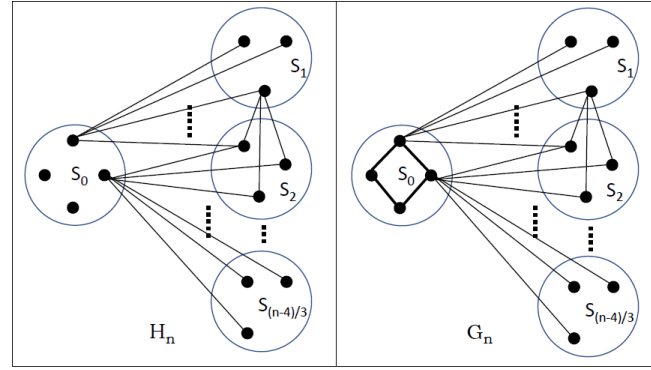

Fig. 3: On the left is H_n where each vertex v in S_i is connected to each vertex u in S_j, $i \neq j$. On the right is G_n which is formed from H_n by adding four edges to S_0. For the case $(n \mod 3) = 1$, H_n and G_n are non-isomorphic graphs on n vertices, with $f(n)$ maximal cliques each, showing a counterexample to Theorem 2 of Moon and Moser [29].

except that the vertices within S_0 are connected by a cycle of length 4. In this case, we can still construct $4 \cdot 3^{\frac{n-4}{3}}$ maximal cliques, since we can make a maximal clique by choosing two connected vertices in S_0 (4 ways to do this), and one vertex from each $S_i, i > 0$ (3 ways for each such $S_i, i = 1 \ldots (n-4)/3$).

Observation 1 *For the case $(n \mod 3) = 1$, there are two distinct non-isomorphic graphs H_n and G_n described above, that have $4 \cdot 3^{\frac{n-4}{3}}$ maximal cliques, which is the maximum possible. This is a correction to Theorem 2 of Moon and Moser [29], which states that there is only one such graph, H_n.*

This observation enables us to have $\lambda(n) = 2f(n)$ for the case $(n \mod 3) = 1$. By starting with graph H_n and by adding edges to make it G_n, we remove $f(n)$ maximal cliques and introduce $f(n)$ maximal cliques, leading to a total change of $2f(n)$.

4 Enumeration of Change in Set of Maximal Cliques

In this section we present algorithms for enumerating the change in the set of maximal cliques. In Section 4.1, we first present an algorithm with provable theoretical properties for enumerating new maximal cliques that arise due to the addition of a batch of edges, followed by an algorithm with good practical performance in Section 4.2. In Section 4.3, we present an algorithm for enumerating subsumed cliques due to the addition of new edges. We then consider the decremental case where edges are deleted from the graph in Section 4.4. For graph G and edge set H, when the context is clear, we use Λ^{new} to mean $\Lambda^{new}(G, G+H)$ and similarly Λ^{del} to mean $\Lambda^{del}(G, G+H)$.

4.1 Enumeration of New Maximal Cliques When Edge Set H is Added

When a set of edges H is added to the graph G, let G' denote the graph $G + H$. One approach to enumerating new cliques in G' is to simply enumerate all cliques in G' using an output-sensitive algorithm such as [7], suppress those cliques that were also present in G, and output the rest. However, this is clearly not change-sensitive, since it is possible that most cliques in G' will not be finally output. An important point to note here is that most similar approaches, that involve enumerating maximal cliques in a certain graph, followed by suppressing cliques that do not belong to Λ^{new}, run this risk of sometimes having to suppress most of the cliques that were enumerated, and such approaches will not be change-sensitive. In the following, we present a simple approach, which, at its core, directly outputs cliques from Λ^{new}, and does not output cliques that do not belong to Λ^{new}.

For edge $e \in H$, let $C'(e)$ denote the set of maximal cliques in G' that contain edge e. We first present the following observation that Λ^{new}, the set of all new maximal cliques, is precisely the set of all maximal cliques in G' that contain at least one edge from H.

Lemma 1

$$\Lambda^{new}(G, G') = \cup_{e \in H} C'(e)$$

Proof We first note that each clique in Λ^{new} must contain at least one edge from H. We use proof by contradiction. Consider a clique $c \in \Lambda^{new}$. If c does not contain an edge from H, then c is also a clique in G, and hence cannot belong to Λ^{new}. Hence, $c \in C'(e)$ for some edge $e \in H$, and $c \in \cup_{e \in H} C'(e)$. This shows that $\Lambda^{new} \subseteq \cup_{e \in H} C'(e)$. Next consider a clique $c \in \cup_{e \in H} C'(e)$. It must be the case that $c \in C'(h)$ for some h in H. Thus c is a maximal clique in G'. Since c contains edge $h \in H$, c cannot be a clique in G. Thus $c \in \Lambda^{new}$. This shows that $\cup_{e \in H} C'(e) \subseteq \Lambda^{new}$. □

We now consider efficient ways of enumerating cliques from $\cup_{e \in H} C'(e)$. For an edge $e \in H$, the enumeration of cliques in $C'(e)$ is reduced to the enumeration of *all* maximal cliques in a specific subgraph of G', as follows. Let u and v denote the endpoints of e, and let G'_e denote the induced subgraph of G' on the vertex set $\{u, v\} \cup \{\Gamma_{G'}(u) \cap \Gamma_{G'}(v)\}$ i.e. the set of vertices adjacent to both u and v in G', in addition to u and v. For example, see Figure 4 for construction of G'_e.

Lemma 2

For each $e \in H$, $C'(e) = \mathcal{C}(G'_e)$

Proof First we show that $C'(e) \subseteq \mathcal{C}(G'_e)$. Consider a clique c in $C'(e)$, i.e. a maximal clique in $G' = G + H$ containing

edge e. Hence c must contain both u and v. Every vertex in c (other than u and v) must be connected to both u and to v in G', and hence must be in $\Gamma_{G'}(u) \cap \Gamma_{G'}(v)$. Hence c must be a clique in G'_e. Since c is a maximal clique in G', and G'_e is a subgraph of G', c must also be a maximal clique in G'_e. Hence we have that $c \in \mathcal{C}(G'_e)$, leading to $C'(e) \subseteq \mathcal{C}(G'_e)$.

Next, we show that $\mathcal{C}(G'_e) \subseteq C'(e)$. Consider any maximal clique d in G'_e. We note the following in G'_e: (1) every vertex in G'_e (other than u and v) is connected to u as well as v (2) u and v are connected to each other. Due to these conditions, d must contain both u and v, and hence also edge $e = (u, v)$. Clearly, d is a clique in G' that contains edge e. We now show that d is a maximal clique in G'. Suppose not, and we could add vertex v' to d and it remained a clique in G'. Then, v' must be in $\Gamma_{G'}(u) \cap \Gamma_{G'}(v)$, and hence v' must be in G'_e, so that d is not a maximal clique in G'_e, which is a contradiction. Hence, it must be that d is a maximal clique in G' that contains edge e, and $d \in C'(e)$. □

Following Lemma 2, in Figure 4, $\{2, 3, 4, 5\}$ is a new maximal clique in G' that contains $e = (4, 5) \in H, H = \{(3, 5), (4, 5)\}$. Note that $\{2, 3, 4, 5\}$ is also a maximal clique in G'_e.

Our change-sensitive algorithm, IMCENewClq (Algorithm 1) is based on the above observation, and uses an output-sensitive algorithm MCE, due to [7], to enumerate all maximal cliques in G'_e.

Algorithm 1: IMCENewClq(G, H)

Input: G - Input graph, H - Set of ρ edges added to G
Output: All cliques in Λ^{new}, each clique output once
1 Consider edges of H in an arbitrary order $e_1, e_2, \ldots, e_\rho$
2 $G' \leftarrow G + H$
3 **for** $i = 1 \ldots \rho$ **do**
4      $e \leftarrow e_i = (u, v)$
5      $V_e \leftarrow \{u, v\} \cup \{\Gamma_{G'}(u) \cap \Gamma_{G'}(v)\}$
6      $G'_e \leftarrow$ graph induced by V_e on G'
7   Generate cliques using MCE(G'_e). For each clique c thus generated, output c only if c does not contain an edge e_j for $j < i$

Theorem 4 IMCENewClq *enumerates the set of all new cliques arising from the addition of H in time $O(\Delta^3 \rho | \Lambda^{new} |)$ where Δ is the maximum degree of a vertex in G'. The space complexity is $O(|E(G + H)| + |V(G + H)|)$.*

Proof We first prove the correctness of the algorithm. From Lemmas 1 and 2, we have that by enumerating $\mathcal{C}(G'_e)$ for every $e \in H$, we enumerate Λ^{new}. Our algorithm does exactly that, and enumerates $\mathcal{C}(G'_e)$ using Algorithm MCE. Note that each clique $c \in \Lambda^{new}$ is output exactly once though c maybe in $\mathcal{C}(G'_e)$ for multiple edges $e \in H$. This is because c is output only for edge e that occurs earliest in the pre-determined ordering of edges in H.
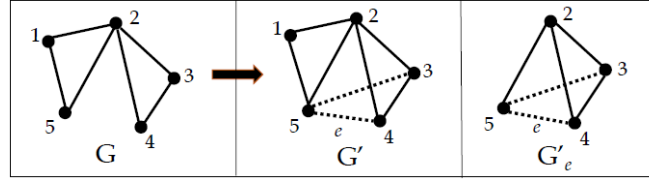
Fig. 4: Illustration of Lemma 2 that, the set of new maximal cliques in G' containing $e = (4,5)$, i.e. the single clique $\{2,3,4,5\}$, is exactly the set of all maximal cliques in G'_e.

For the runtime, consider that the algorithm iterates over the edges in H. In an iteration involving edge e, it constructs a graph G'_e and runs $\texttt{MCE}(G'_e)$. Note that the number of vertices in G'_e is no more than $\Delta + 1$, and is typically much smaller, since it is the size of the intersection of two vertex neighborhoods in G'. Since the arboricity of a graph is less than its maximum degree, $\alpha' \leq \Delta$ where α' is the arboricity of G'_e. Further, the number of edges in G'_e is $O(\Delta^2)$. The set of maximal cliques generated in each iteration is a subset of Λ^{new}, hence the number of maximal cliques generated from each iteration is no more than $|\Lambda^{new}|$. Applying Theorem 2, we have that the runtime of each iteration is $O(\Delta^3|\Lambda^{new}|)$. Since there are ρ iterations, the result on runtime follows.

For the space complexity, we note that the algorithm does not store the set of new cliques in memory at any point. The space required to construct G'_e is linear in the size of $G' = (G + H)$, and so is the space requirement of Algorithm $\texttt{MCE}(G'_e)$, from Theorem 2. Hence the total space requirement is linear in the number of edges in $G + H$. $\quad\square$

4.2 Practical Algorithm for Enumerating New Maximal Cliques

The algorithm $\texttt{IMCENewClq}$ uses as a subroutine Algorithm $\texttt{MCE}$ (Chiba and Nishizeki [7]) to enumerate maximal cliques within a subgraph of G. While $\texttt{MCE}$ has theoretical properties of being output-sensitive, in practice, it is not the fastest algorithm for maximal clique enumeration. In practice, the most efficient algorithms for maximal clique enumeration in a static graph are based on depth-first search using a technique called "pivoting", such as the algorithm due to Tomita et al. [41]. While the runtime of these algorithms are not provably output-sensitive, they are faster in practice than those algorithms that are provably output-sensitive.

The algorithm due to Tomita et al. [41], which we call $\texttt{TTT}$, is a recursive algorithm based on backtracking for enumerating $\mathcal{C}(G)$, given G. In its recursive procedure, it maintains a currently found clique, not necessarily maximal, and adds vertices one to the current clique, declaring the current clique to be maximal when no further vertices can be added. The vertices are considered in a carefully chosen order using a method called "pivoting". $\texttt{TTT}$ is shown to be worst-case optimal with a runtime of $O(3^{n/3})$ for an n vertex graph. It

is possible to improve the performance of the $\texttt{IMCENewClq}$ algorithm by directly using $\texttt{TTT}$ in place of $\texttt{MCE}$. In the following, we show how to do even better.

*Reducing Redundant Clique Computation:*Note that $\texttt{IMCENewClq}(G,H)$ may compute the same clique c multiple times, for example, if $c \in C'(e_1)$ and $c \in C'(e_2)$ for $e_1 \neq e_2$. Duplicates are suppressed prior to emitting the cliques, by outputting c only for one of the edges among $\{e_1, e_2\}$, but the algorithm still pays the computational cost of computing a clique such as c multiple times. We present Algorithm $\texttt{FastIMCENewClq}$ that eliminates the cost of having to even compute a clique such as c multiple times.

Algorithm 2: $\texttt{TTTExcludeEdges}(\mathcal{G}, K, \texttt{cand}, \texttt{fini}, \mathcal{E})$

Input: $\mathcal{G}$ - The input graph, K - a non-maximal clique to extend
$\quad$ $\texttt{cand}$ - Set of vertices that may extend K, $\texttt{fini}$ - vertices that have been used to extend K
$\quad$ $\mathcal{E}$ - set of edges to exclude

1 **if** $(\texttt{cand} = \emptyset)$ & $(\texttt{fini} = \emptyset)$ **then**
2 $\quad$ Output K and return

3 $\texttt{pivot} \leftarrow (u \in \texttt{cand} \cup \texttt{fini})$ such that u maximizes the size of $\texttt{cand} \cap \Gamma_{\mathcal{G}}(u)$
4 $\texttt{ext} \leftarrow \texttt{cand} - \Gamma_{\mathcal{G}}(\texttt{pivot})$
5 **for** $q \in \texttt{ext}$ **do**
6 $\quad$ $K_q \leftarrow K \cup \{q\}$
7 $\quad$ **if** $K_q \cap \mathcal{E} \neq \emptyset$ **then**
8 $\quad\quad$ $\texttt{cand} \leftarrow \texttt{cand} - \{q\}$; $\texttt{fini} \leftarrow \texttt{fini} \cup \{q\}$
9 $\quad\quad$ continue
10 $\quad$ $\texttt{cand}_q \leftarrow \texttt{cand} \cap \Gamma_{\mathcal{G}}(q)$; $\texttt{fini}_q \leftarrow \texttt{fini} \cap \Gamma_{\mathcal{G}}(q)$
11 $\quad$ $\texttt{TTTExcludeEdges}(\mathcal{G}, K_q, \texttt{cand}_q, \texttt{fini}_q, \mathcal{E})$
12 $\quad$ $\texttt{cand} \leftarrow \texttt{cand} - \{q\}$; $\texttt{fini} \leftarrow \texttt{fini} \cup \{q\}$

Algorithm $\texttt{FastIMCENewClq}$ uses as a subroutine Algorithm $\texttt{TTTExcludeEdges}$, an extension of the $\texttt{TTT}$ algorithm, which enumerates all maximal cliques of an input graph that avoid a given set of edges. While $\texttt{TTT}$ simply takes a graph as input and enumerates all maximal cliques within the graph, $\texttt{TTTExcludeEdges}$ takes an additional input, a set of edges $\mathcal{E}$, and only enumerates those cliques within the graph that do not contain any edge from $\mathcal{E}$. We present the recursive version of $\texttt{TTTExcludeEdges}$, which takes as input five parameters – an input graph G, three sets

---

Algorithm 3: $\texttt{FastIMCENewClq}(G, H)$

Input: G - input graph
         H - Set of ρ edges being added to G
Output: Cliques in $\Lambda^{new} = \mathcal{C}(G + H) \setminus \mathcal{C}(G)$

1   $G' \leftarrow G + H$; $\mathcal{E} \leftarrow \phi$
2   Consider edges of H in an arbitrary order $e_1, e_2, \ldots, e_\rho$
3   **for** $i \leftarrow 1, 2, \ldots, \rho$ **do**
4      $e \leftarrow e_i = (u, v)$
5      $V_e \leftarrow \{u, v\} \cup \{\Gamma_{G'}(u) \cap \Gamma_{G'}(v)\}$
6      $\mathcal{G} \leftarrow$ Graph induced by V_e on G'
7      $K \leftarrow \{u, v\}$
8      $\texttt{cand} \leftarrow V_e \setminus \{u, v\}$; $\texttt{fini} \leftarrow \emptyset$
9      $S \leftarrow \texttt{TTTExcludeEdges}(\mathcal{G}, K, \texttt{cand}, \texttt{fini}, \mathcal{E})$
10     $\Lambda^{new} \leftarrow \Lambda^{new} \cup S$
11     $\mathcal{E} \leftarrow \mathcal{E} \cup e_i$

---

of vertices K, $\texttt{cand}$, and $\texttt{fini}$, and a set of edges $\mathcal{E}$. The algorithm outputs every maximal clique in G that (a) contain all vertices in K, (b) zero or more vertices in $\texttt{cand}$, (c) none of the vertices in $\texttt{fini}$, and (d) none of the edges in $\mathcal{E}$.

A description of $\texttt{TTTExcludeEdges}$ is presented in Algorithm 2. We note that this algorithm follows the structure of the recursion in the TTT algorithm, and incorporates additional pruning of search paths, by avoiding paths that contain an edge from $\mathcal{E}$. In particular, in Line 7 of $\texttt{TTTExcludeEdges}$, if the clique K_q (formed after adding vertex q to K) contains an edge from $\mathcal{E}$, then the rest of the search path, which will continue adding more vertices, is not explored further. Instead the algorithm backtracks and tries to extend the clique K by adding other vertices.

Our algorithm for enumerating new maximal cliques $\texttt{FastIMCENewClq}$ (Algorithm 3) is an adaptation of change-sensitive algorithm $\texttt{IMCENewClq}$ (Algorithm 1) where we use $\texttt{TTTExcludeEdges}$ instead of the output-sensitive MCE. In particular, while enumerating all new cliques containing edge e_i, $\texttt{FastIMCENewClq}$ enumerates only those cliques that exclude edges $\{e_1, e_2 \ldots, e_{i-1}\}$. Note that in $\texttt{FastIMCENewClq}$, there is no further duplicate suppression required, since the call to $\texttt{TTTExcludeEdges}$ does not return any cliques that contain an edge from $\mathcal{E}$. This is more efficient than first enumerating duplicate cliques, followed by suppressing duplicates before emitting them. This idea makes $\texttt{FastIMCENewClq}$ more efficient in practice than $\texttt{IMCENewClq}$.

The correctness of $\texttt{FastIMCENewClq}$ follows in a similar fashion to that of Algorithm $\texttt{IMCENewClq}$ proved in Theorem 4, except that we also need a proof of the guarantee provided by Algorithm $\texttt{TTTExcludeEdges}$, which we prove below.

Lemma 3 $\texttt{TTTExcludeEdges}(\mathcal{G}, K, \texttt{cand}, \texttt{fini}, \mathcal{E})$ *(Algorithm 2) returns all maximal cliques c in $\mathcal{G}$ such that (1) c contains all vertices from K, (2) remaining vertices in c are chosen from $\texttt{cand}$, (3) c contains no vertex from $\texttt{fini}$ and (4) c does not contain any edges in $\mathcal{E}$.*

Proof We note that $\texttt{TTTExcludeEdges}$ matches the original TTT algorithm, except for lines 7 to 9. Hence, if we do not consider lines 7 to 9 in $\texttt{TTTExcludeEdges}$, the algorithm becomes TTT, and by the correctness of TTT (Theorem 1 [41]), all maximal cliques c in $\mathcal{G}$ are returned.

Now consider lines 7 to 9 in $\texttt{TTTExcludeEdges}$. Clearly, (1), (2), (3) are preserved for each maximal clique c generated by $\texttt{TTTExcludeEdges}$. Now to complete the correctness proof of $\texttt{TTTExcludeEdges}$, along with proving (4), we also need to prove that each maximal clique c in $\mathcal{G}$ that does not contain any edge in $\mathcal{E}$ is generated by $\texttt{TTTExcludeEdges}$.

Assume there exists a maximal clique c in $\mathcal{G}$, which contains an edge in $\mathcal{E}$, which is output by the $\texttt{TTTExcludeEdges}$ algorithm, assume the offending edge is $e = (q, v)$. Suppose that vertex v was added to our expanding clique first. Then, as q is processed, line 7 of the algorithm will return back true as $e \in K_q$ and $\mathcal{E}$, thus q will not be added to the clique, and c will not be reported as maximal, a contradiction.

Now we will show that, if a maximal clique does not contain an edge from $\mathcal{E}$, the clique will be generated. Consider a maximal clique c in $\mathcal{G}$ that contains no edge from $\mathcal{E}$ but c is not generated by $\texttt{TTTExcludeEdges}$. The only reason for c not being generated is the inclusion of lines 7 to 9 (of $\texttt{TTTExcludeEdges}$) to TTT resulting in $\texttt{TTTExcludeEdges}$, because, otherwise, c would be generated due to correctness of TTT. So in $\texttt{TTTExcludeEdges}$, during the expansion of K towards c, there exists a vertex $q \in c$ such that line 7 in $\texttt{TTTExcludeEdges}$ is satisfied and c never gets a chance to be generated as q is excluded from $\texttt{cand}$ and included in $\texttt{fini}$ (line 8). This implies that c contains at least an edge in $\mathcal{E}$, because otherwise, condition at line 7 would never be satisfied. This is a contradiction. Hence, c must be generated. This completes the proof. $\square$

4.3 Enumeration of Subsumed Maximal Cliques

We now consider the enumeration of subsumed cliques, i.e. the set $\mathcal{C}(G) \setminus \mathcal{C}(G + H)$. A subsumed clique c' still exists in $G' = G + H$, but is now a part of a larger clique in G'. Such a larger clique must be a part of Λ^{new}. Thus, an algorithm idea is to check each new clique c in Λ^{new} to see if c subsumed any maximal clique c' in G. In order to see which maximal cliques c' may have subsumed, we note that any maximal clique subsumed by c must also be a maximal clique within subgraph $c - H$. Thus, one approach is to enumerate all maximal cliques in $c - H$ and for each such generated clique c', we check whether c' is maximal in G by verifying maximality of c' in G. This algorithm can be implemented in space proportional to the size of $G + H$, since it can directly use an algorithm for maximal clique enumeration such as MCE.
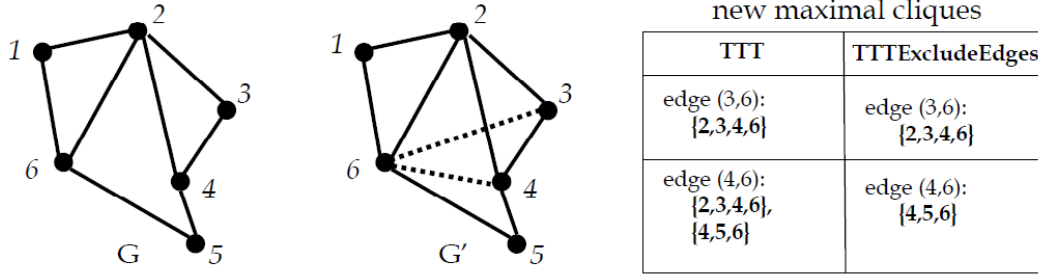
new maximal cliques	
TTT	**TTTExcludeEdges**
edge (3,6): {2,3,4,6}	edge (3,6): {2,3,4,6}
edge (4,6): {2,3,4,6}, {4,5,6}	edge (4,6): {4,5,6}

Fig. 5: Enumeration of new maximal cliques when graph changes from G to G' due to addition of new edges $(3,6)$ and $(4,6)$. Consider the ordering of edges as $(3,6)$ followed by $(4,6)$. There are two new maximal cliques containing edge $(4,6)$, which are $\{4,5,6\}$ and $\{2,3,4,6\}$. With `TTTExcludeEdges`, only $\{4,5,6\}$ is enumerated when considering edge $(4,6)$, since $\{2,3,4,6\}$ has already been enumerated while considering edge $(3,6)$.

However, in practice, checking each potential clique for maximality is a costly operation since it potentially needs to consider the neighborhood of every vertex of the clique. An alternative approach to avoid this costly maximality check is to store the set of maximal cliques $\mathcal{C}(G)$ and check if c' is in $\mathcal{C}(G)$. The downside of this approach is that the space required to store the clique set can be high.

Hence, we considered another approach to subsumed cliques, where we reduce the memory cost by storing signatures of maximal cliques as opposed to the cliques themselves. The signature is computed by representing a clique in a canonical fashion (for instance, representing the clique as a list of vertices sorted by their ids.) as a string followed by computing a hash of this string. By storing only the signatures and not the cliques themselves, we are able to check if a clique is a current maximal clique, and at the same time, pay far lesser cost in memory when compared with storing the clique itself. The algorithm is described in Algorithm 4, With this approach of storing signatures instead of storing the cliques themselves, there is a (small) chance of collision of signatures, which means for two different cliques C_1 and C_2 the signature might be the same. This might result in false positives meaning that some cliques might wrongly be concluded as subsumed cliques. However, the probability of the event that the hash values of two different cliques are same is extremely low with the use of a hashing algorithm such as 64-bit murmur hash [2]. In our experiments, we observed that the set of subsumed cliques reported with the use of signature is always the same as the actual set of subsumed cliques. If it is extremely important to avoid false positives, we can explicitly check a potential subsumed clique for maximality in the original graph.

In Algorithm 4, Lines 4 to 12 describes the procedure for computing S, the set of all maximal cliques in $c - H$ and Lines 13 to 15 decide which among the maximal cliques in S are subsumed. For computing maximal cliques in $c - H$,

[2] https://sites.google.com/site/murmurhash/

we only consider the edges in H that are present in c as we can see in Line 4. We prove that S is the set of all maximal cliques in $c - H$ in the following lemma:

Lemma 4 *In Algorithm 4, for each $c \in \Lambda^{new}$, S contains all maximal cliques in $c - H$.*

Proof First note that, we only consider the set of all edges $H_1 \subseteq H$ which are present in c (line 4). Clearly computing maximal cliques in $c - H$ reduces to computing maximal cliques in $c - H_1$. We prove this using induction on k, the number of edges in H_1. Suppose $k = 1$ so that H_1 is a single edge, say $e_1 = \{u, v\}$. Then clearly, $c - H_1$ has two maximal cliques, $c \setminus \{u\}$ and $c \setminus \{v\}$, proving the base case. Suppose that for any set H_1 of size k, it is true that all maximal cliques in $c - H_1$ have been generated using induction hypothesis. Consider a set $H_1' = \{e_1, e_2, ..., e_{k+1}\}$ with $(k+1)$ edges. Now each maximal clique c' in $c - H_1$ either remains a maximal clique within $c - H_1'$ (if at least one endpoint of e_{k+1} is not in c'), or leads to two maximal cliques in $c - H_1'$ (if both endpoints of e_{k+1} are in c'). Thus lines 4 to 12 in Algorithm 4 generates all maximal cliques in $c - H$. □

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

Lemma 5 *Algorithm `IMCESubClq` (Algorithm 4) enumerates all cliques in $\Lambda^{del} = \mathcal{C}(G) \setminus \mathcal{C}(G')$ using time $O(2^\rho |\Lambda^{new}|)$. The space complexity of the algorithm is $O(|E(G')| + |V(G')| + |\mathcal{C}(G)|)$. The algorithm can also be adapted to run in time $O(2^\rho |E(G)||\Lambda^{new}|)$, and space $O(|E(G')| + |V(G')|$.*

Proof We first show that every clique c' enumerated by the algorithm is indeed a clique in Λ^{del}. To see this, note that c' must be a maximal clique in G, due to explicitly checking the condition. Further, c' is not a maximal clique in G', since it is a proper subgraph of c, a maximal clique in G'. Next, we

---

Algorithm 4: `IMCESubClq`(G, H, C, Λ^{new})

Input: G - Input Graph
        H - Edge set being added to G
        C - Set of maximal cliques in G
        Λ^{new} - set of new maximal cliques in $G + H$
Output: All cliques in $\Lambda^{del} = \mathcal{C}(G) \setminus \mathcal{C}(G + H)$

1 $\Lambda^{del} \leftarrow \emptyset$
2 **for** $c \in \Lambda^{new}$ **do**
3     $S \leftarrow \{c\}$
4   **for** $e = (u,v) \in E(c) \cap H$ **do**
5        $S' \leftarrow \phi$
6   **for** $c' \in S$ **do**
7   **if** $e \in E(c')$ **then**
8              $c_1 = c' \setminus \{u\}$; $c_2 = c' \setminus \{v\}$
9              $S' \leftarrow S' \cup c_1$; $S' \leftarrow S' \cup c_2$
10   **else**
11              $S' \leftarrow S' \cup c'$
12        $S \leftarrow S'$
13   **for** $c' \in S$ **do**
14   **if** $c' \in C$ **then**
15           $\Lambda^{del} \leftarrow \Lambda^{del} \cup c'$
16           $C \leftarrow C \setminus c'$

Algorithm 5: `Decremental`(G, H)

Input: G - Input Graph, H - Set of ρ edges being deleted
Output: All cliques in $\Lambda^{new}(G, G - H) \cup \Lambda^{del}(G, G - H)$

1 $\Lambda^{new} \leftarrow \emptyset$, $\Lambda^{del} \leftarrow \emptyset$, $G'' \leftarrow G - H$
2 $\Lambda^{del} \leftarrow$ `IMCENewClq`(G'', H)
3 $\Lambda^{new} \leftarrow$ `IMCESubClq`$(G'', H, \mathcal{C}(G''), \Lambda^{del})$

---

show that all cliques in Λ^{del} are enumerated. Consider any subsumed clique $c'_1 \in \Lambda^{del}$. It must be contained within $c_1 - H$, where $c_1 \in \Lambda^{new}$. Moreover, c'_1 will be a maximal clique within $c_1 - H$, and will be enumerated by the algorithm according to Lemma 4.

For the time complexity we show that for any $c \in \Lambda^{new}$, the maximum number of maximal cliques in $c^{-H} = c - H$ is 2^ρ. Proof is by induction on ρ. Suppose $\rho = 1$ so that H is a single edge, say $e_1 = \{u, v\}$. Then clearly c^{-H} has two maximal cliques, $c \setminus \{u\}$ and $c \setminus \{v\}$, proving the base case. Suppose that for any set H of size k, it was true that c^{-H} has no more than 2^k maximal cliques. 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 $c - H''$ is obtained from $c - H'$ by deleting a single edge e_{k+1}. By induction, we have that $c - H'$ has no more than 2^k maximal cliques. Each maximal clique c' in $c - H'$ either remains a maximal clique within $c - H''$ (if at least one endpoint of e_{k+1} is not in c') , or leads to two maximal cliques in $c - H''$ (if both endpoints of e_{k+1} are in c'). Hence, the number of maximal cliques in $c - H''$ is no more than 2^{k+1}, completing the inductive step.

Thus, for each cliques $c \in \Lambda^{new}$, we need to check maximality for no more than 2^ρ cliques in G. Note that a clique c' is maximal in G if it is contained in $\mathcal{C}(G)$, the set of maximal cliques in G. This can be done in constant time by storing the signatures of maximal cliques and checking if the signature of c' is in the set of signatures of maximal cliques of G.

For the space bound, we first note that all operations in Algorithm 4 except maximality check can be done in space

linear in the size of G'. For maximality check we need space $O(|\mathcal{C}(G)|)$ as we need to store the (signatures of) maximal cliques of G. The only remaining space cost is the size of Λ^{new}, which can be large. Note that the algorithm only iterates through Λ^{new} in a single pass. If elements of Λ^{new} were provided as a stream from the output of an algorithm such as `IMCENewClq`, then they do not need to be stored within a container, so that the memory cost of receiving Λ^{new} is reduced to the cost of storing a single maximal clique within Λ^{new} at a time.

An alternative algorithm does not store $\mathcal{C}(G)$ (or hashes of elements in $\mathcal{C}(G)$). Instead, each time a potential subsumed clique c' is generated that is contained in a new clique $c \in \Lambda^{new}$, we simply check c' for maximality in G. This can be done in time $O(|E(G)|)$, by checking the intersections of the different vertex neighborhoods – typical runtime for maximality checking can be much smaller. □

4.4 Decremental Case

We next consider the case when a set of edges H is deleted from G, as opposed to added to G. We start from graph G and go to graph $G - H$, and we are interested in efficiently enumerating $\Lambda(G, G - H)$. The decremental case can be reduced to the incremental case through the following observation.

Observation 2 $\Lambda^{del}(G, G - H) = \Lambda^{new}(G - H, G)$ and $\Lambda^{new}(G, G - H) = \Lambda^{del}(G - H, G)$

Proof Consider the first equation: $\Lambda^{del}(G, G - H) = \Lambda^{new}(G - H, G)$. Let $c \in \Lambda^{del}(G, G - H)$. This means that $c \in \mathcal{C}(G)$ and $c \notin \mathcal{C}(G - H)$. Equivalently, c is not a maximal clique in $G - H$, but upon adding H to $G - H$, c becomes a maximal clique in G. Hence, it is equivalent to say that $c \in \Lambda^{new}(G - H, G)$. Hence, we have $\Lambda^{del}(G, G - H) = \Lambda^{new}(G - H, G)$. The other equation, $\Lambda^{new}(G, G - H) = \Lambda^{del}(G - H, G)$, can be proved similarly. □

The decremental case is outlined in Algorithm 5.

Fully Dynamic Case: Consider the *fully dynamic case*, where there is a set of insertions (edge set H) as well as deletions (edge set H') from a graph. This can be processed as follows. First, we ensure there is no overlap between H

and H', i.e. $H \cap H' = \emptyset$. If this is not the case, we can simply remove overlapping elements since they have no effect on the final graph. Next, we enumerate the change following all the edge deletions, followed by enumerating the change upon edge insertions. Note however, that this may not lead to a change-sensitive algorithm. Intermediate cliques that are output may not be in the final set of new or subsumed cliques.

5 Experimental Evaluation

In this section, we present results from empirical evaluation of the performance of algorithms proposed in this paper. We address the following questions: (1) What is the runtime and memory usage for maintaining the set of maximal cliques of a dynamic graph? (2) How does the runtime compare with the magnitude of the change? (3) How do our algorithms compare with prior work?

5.1 Datasets

Real Dynamic Graphs: We consider graphs from the Stanford large graph database [24] and KONECT- The Koblenz Network Collection [3]: `dblp-coauthor` is a co-authorship network where each vertex represents an author and there is an edge between two authors if they have a common publication. `flickr-growth` is a social network of Flickr users where each vertex represents a user and there exists a directed edge if two users are friends. `sx-stackoverflow-a2q` is a social network where each vertex represents a user on stackoverflow, and if user a answers user b's question then there is a directed edge from a to b. `wiki-talk` is a network of Wikipedia users where each vertex represents a user and if user a edited user b's talk page then there exists a directed edge from a to b. `wikipedia-growth` is a hyperlink network of the English Wikipedia where each vertex represents a wikipedia page and there is an edge from a page $wiki_1$ to a page $wiki_2$ if there is a hyperlink of $wiki_2$ from $wiki_1$. `youtube-u-growth` is a social network of youtube users, where nodes are the users and there is an edge between two users if they are friends. In each graph, edges have time-stamps of creation. We convert all these graphs into simple undirected graphs. If there are multiple time-stamp edges between two vertices, we take the edge with the earliest time-stamp. A summary of the graphs used in this experiment is given in Table 1. In our experiments, we start with the empty graph and at each iteration, we add a batch of new edges, and enumerate the change resulting after the addition.

Synthetic Graphs: We also considers a variant of the Erdős-Rényi random graph model $G(n, N)$ graph for our experiments where n is the number of vertices and N is the number of edges. In these, we first generate graphs according to the standard Erdős-Rényi random graph model [13], and we "plant" cliques of a certain size. We call these graphs `ER-1M-20M` with 1M vertices, 20M edges and `ER-2M-15M` with 2M vertices and 15M edges. We plant 10 random cliques each of size 20 on `ER-1M-20M` and 10 random cliques each of size 30 on `ER-2M-15M`, with the goal of finding the planted cliques through incremental computation.
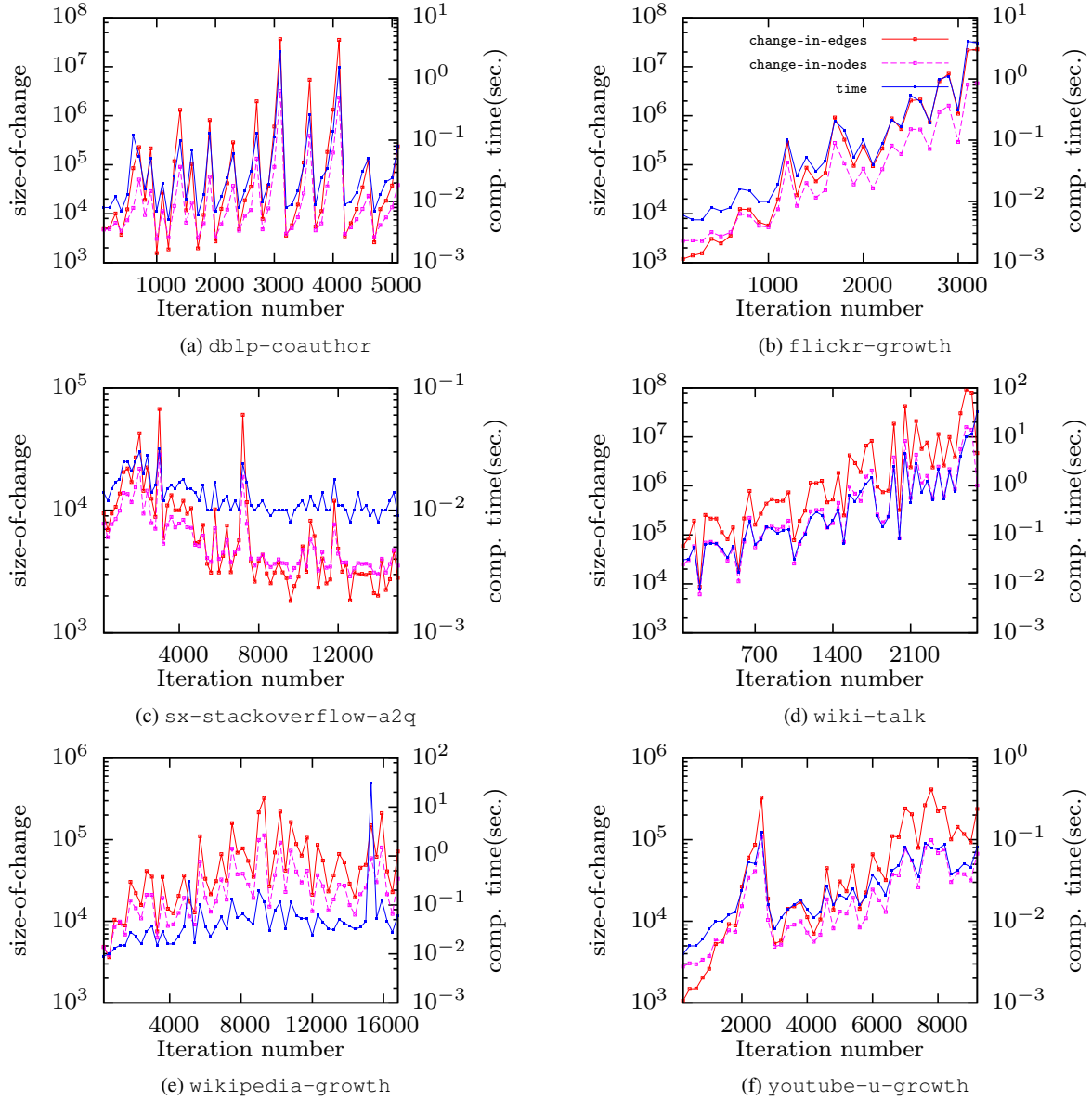
5.2 Experimental Setup and Implementation Details

We implemented all the algorithms in Java on a 64-bit Intel(R) Xeon(R) CPU with 8G DDR3 RAM with 6G heap memory.

Algorithms: We evaluate our algorithm `IMCE` for maintenance of maximal cliques. `IMCE` consists of `FastIMCENewClq` for enumerating new maximal cliques and `IMCESubClq` for enumerating subsumed maximal cliques. We also implemented `IMCENewClq` for enumerating new maximal cliques, but `FastIMCENewClq` performed better, hence we present results for `FastIMCENewClq`.

We consider the following prior algorithms for comparison with `IMCE`: (1) `STIX` (Stix [37]) computes on a dynamic graph by incrementally adding one edge at a time; (2) `OV` (Ottosend and Vomlel [32]) computes on a dynamic graph by incrementally adding a set of edges; (3) `MCMEI` (Sun et al. [38]) computes on a dynamic graph by incrementally adding one edge at a time. For the algorithms (`STIX`, `MCMEI`) that support only single edge additions, we simulate the addition of a batch of edges by inserting the edges one at a time.

Metrics: We evaluate the performance of algorithms through the following metrics: **(1)** total computation time for determining new maximal cliques and subsumed maximal cliques when a batch of new edges is added to the graph; **(2)** change-sensitiveness, i.e, total computation time as a function of the size of the total change. We define the size of the total change in terms of edges denoted as change-in-edges as the cumulative sum of the number of edges in the new and subsumed maximal cliques. For example, if there are two new maximal cliques of sizes 3 and 4, and one subsumed clique of size 2, the number of edges of a new maximal clique of size 3 is $\binom{3}{2} = 3$, the number of edges of another new maximal clique of size 4 is $\binom{4}{2} = 6$, and the number of edges of the subsumed clique of size 2 is $\binom{2}{2} = 1$. Hence, the size of total change is $3 + 6 + 1 = 10$. We also consider the size of change in terms of nodes denoted as change-in-nodes where we compute the cumulative number of nodes of all cliques in the change set; **(3)** memory

Dataset	Nodes	Edges
`dblp-coauthor`	1282468	5179996
`flickr-growth`	2302925	22838276
`sx-stackoverflow-a2q`	2433067	15079969
`wiki-talk`	1094018	2722029
`wikipedia-growth`	1870709	36532531
`youtube-u-growth`	3223585	9375374
`ER-1M-20M`	1000000	20001900
`ER-2M-15M`	2000000	15004350

Table 1: Summary of Graphs Used.

Fig. 6: Computation time for enumerating the change in set of maximal cliques for `IMCE`, **and size-of-change per batch (batch size** $\rho = 1000$**). The left** y **axis shows the size of change and the right** y **axis shows the computation time in seconds.**

Dataset	STIX	OV	MCMEI	IMCE
`dblp-coauthor` (464)	3811	285	7237	**0.1**
`flickr-growth` (251)	3664	277	7255	**0.04**
`sx-stackoverflow-a2q` (190)	4883	232	7316	**0.1**
`wiki-talk` (113)	7284	62	1425	**0.1**
`wikipedia-growth` (305)	3923	283	7190	**0.3**
`youtube-u-growth` (172)	3976	279	7257	**0.1**

Table 2: Comparison of different algorithms showing cumulative time (in sec.). The number of batches for which the cumulative time is computed is in the parenthesis. Batch size ρ is set to 100.

Dataset	IMCENewClq	FastIMCENewClq
`dblp-coauthor`(9602)	6768	19
`flickr-growth`(24860)	7318	115
`sx-stackoverflow-a2q`(150800)	2852	65
`wiki-talk`(12777)	7154	75
`wikipedia-growth` (26795)	562	33
`youtube-u-growth` (59814)	2981	82

Table 3: Cumulative computation time (in sec.) for new maximal cliques with batch size $\rho = 100$. The number of batches for which the cumulative time is computed is in the parenthesis.

Dataset	IMCE	IMCENewClq
`ER-1M-20M`	19 min.	24 min.
`ER-2M-15M`	15 min.	15 min.

Table 4: Total time taken to find all the planted cliques incrementally ($\rho = 100$). Other algorithms (`STIX`, `OV`, `MCMEI`) cannot find a single planted clique even in an hour.

cost, which includes the space required to store the graph as well as additional data structures used by the algorithm; and **(4)** cumulative computation time (through a series of incremental updates) as a function of the size of the batch.

5.3 Discussion of Experimental Results

Computation time: Figure 6 shows the computation time of `IMCE` for computing the change in the set of maximal cliques when batches of edges are added. The batch size is set to $\rho = 1000$. On the left y-axis is shown the size of the change, and on the right y-axis is the time for computing the change. We see that the runtime for computing the change in cliques becomes greater as iterations progress for graphs `flickr-growth`, `wiki-talk`, `youtube-u-growth`, and remains roughly the same for other graphs. Fig. 7 shows the breakdown of computation time of `IMCE` into computation time for new maximal cliques (`FastIMCENewClq`) and computation time for subsumed maximal cliques (`IMCESubClq`).

We also compare the computation time of `IMCE` with prior works as shown in Table 2. Clearly, `IMCE` is many orders of magnitude (more than 1000) faster than prior algorithms. One reason why `IMCE` is so much faster than prior works is that `IMCE` systematically selects a local subgraph of the entire graph to search for new and subsumed maximal cliques. This reduces the computation effort considerably. `OV` tried to achieve such a local computation but `OV` is not provably change-sensitive for new maximal cliques, and its computation of subsumed cliques is expensive since the algorithm iterates over the entire set of maximal cliques for deriving subsumed cliques. A similar strategy of iterating over the entire set of maximal cliques for deriving maximal clique set of the updated graph as in `MCMEI` makes the algorithm less efficient.

Next, we compare the computation times of `IMCENewClq` and `FastIMCENewClq` as shown in Table 3. We observe that `FastIMCENewClq` is much faster than `IMCENewClq`. The increase in speed of `FastIMCENewClq` over `IMCENewClq` can be attributed to the additional pruning performed in `FastIMCENewClq` using `TTTExcludeEdges`, when compared with `IMCENewClq` which may enumerate the same clique multiple times (though suppressing it in the output).

On synthetic graphs, we observe that `IMCE` can find all "planted" cliques in approximately 20 min. where as the other algorithms (`STIX`, `OV`, `MCMEI`) could not find a single planted clique in an hour. Results are shown in Table 4.

Change-Sensitiveness: The change in the enumeration time as a function of the size of change can be seen

(a) `dblp-coauthor`

(b) `flickr-growth`

(c) `sx-stackoverflow-a2q`

(d) `wiki-talk`
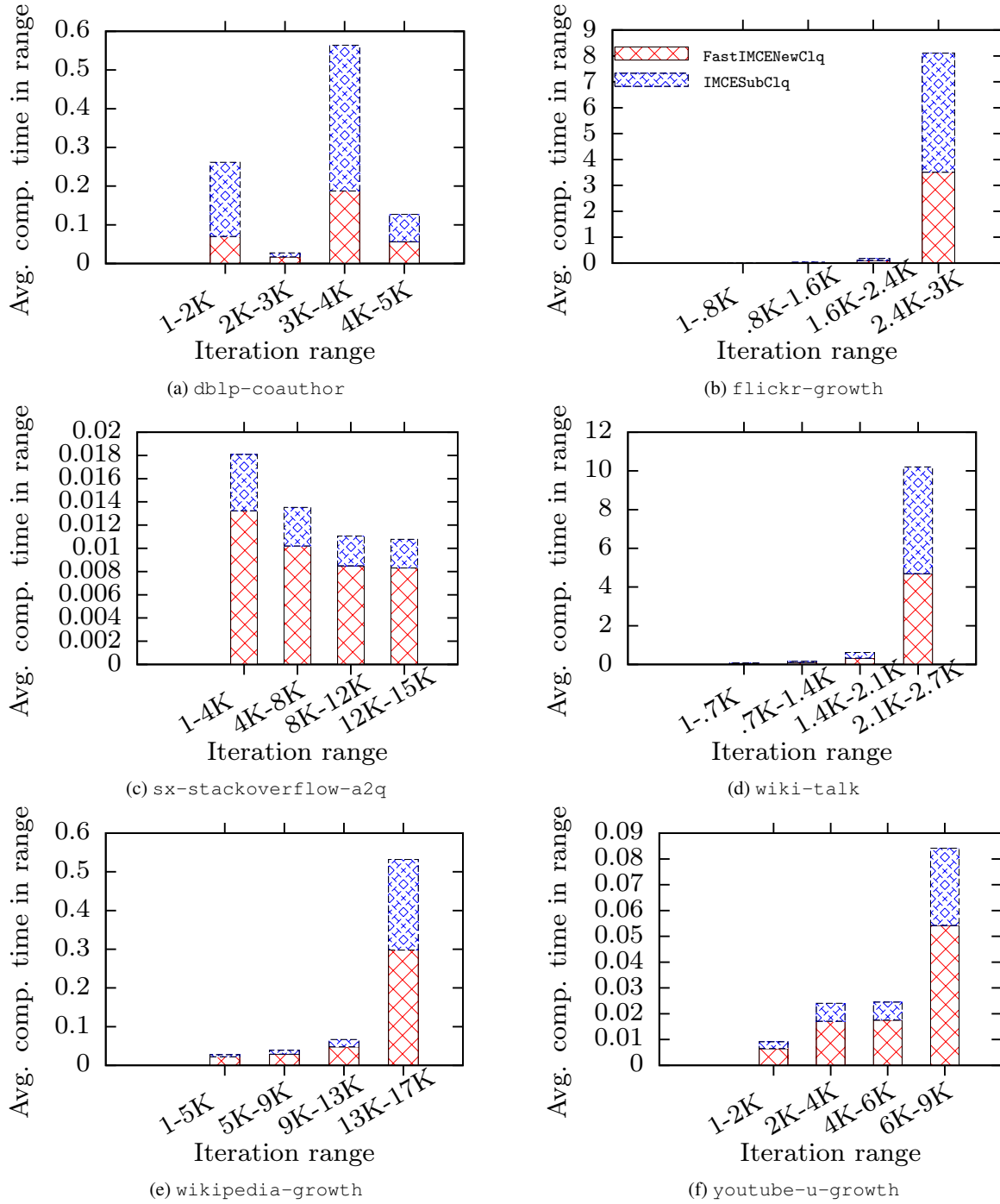
(e) `wikipedia-growth`

(f) `youtube-u-growth`

Fig. 7: Computation time (in sec.) broken down into time for new and subsumed cliques with batch size $\rho = 1000$. Average time in the y-axis is the average taken over the total computation times (new + subsumed) of the iterations in each of the ranges on the x-axis.

in Figure 6. As the iterations progress, the size of change per batch of new edges increases for most graphs, but not in a smooth manner. In general, we observe that the time to compute the change tracks the size of change quite closely, except for graphs `sx-stackoverflow-a2q`, `wikipedia-growth`, `youtube-u-growth`.

For these three graphs, we examined the breakdown of runtime more closely. The time for handling a set of edges consists of three components: (1) graph update time; (2) subgraph computation time (isolating the subgraph for computing new maximal cliques); (3) computation of new and subsumed cliques. Note that the time for the first two com-
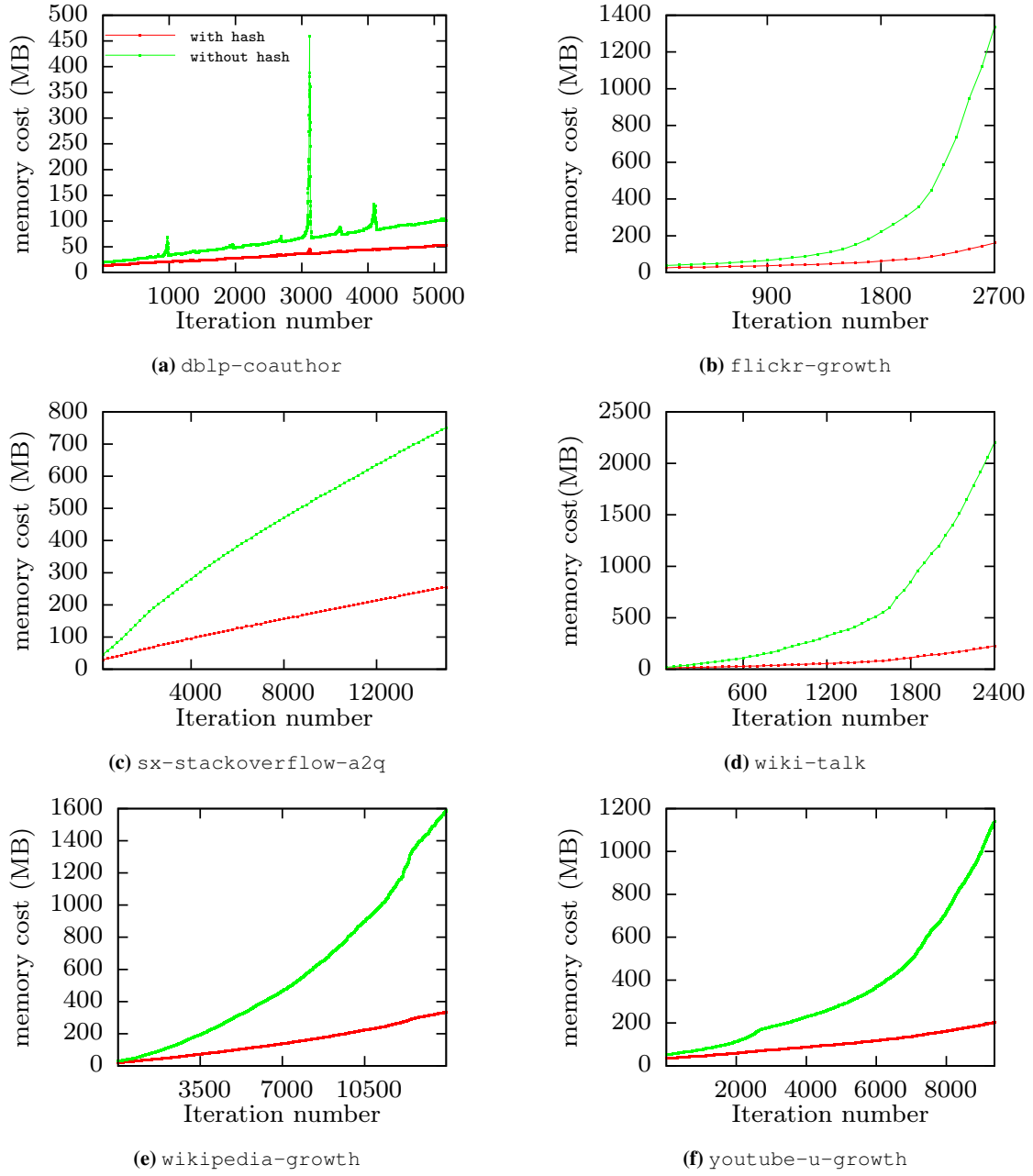
(a) `dblp-coauthor`

(b) `flickr-growth`

(c) `sx-stackoverflow-a2q`

(d) `wiki-talk`

(e) `wikipedia-growth`

(f) `youtube-u-growth`

Fig. 8: Memory cost of `IMCE` with and without using hash function ($\rho = 1000$).

Dataset	$\rho = 1$	$\rho = 10$	$\rho = 100$	$\rho = 1000$	$\rho = 3\log_2 \Delta$
`dblp-coauthor` (5179996)	1622	1317	1166	1264	1204
`flickr-growth` (3298×10^3)	7151	7125	5551	7177	7000
`sx-stackoverflow-a2q` (15079969)	270	270	166	204	106
`wiki-talk` (2717×10^3)	6572	6873	5795	6897	5722
`wikipedia-growth` (17000×10^3)	8869	9093	9134	8495	8678
`youtube-u-growth` (9375374)	459	462	494	400	493

Table 5: Cumulative computation time (in sec.) of `IMCE` with different batch sizes. Note that Δ is the maximum degree of the graph before update. Numbers in the parenthesis indicates the total number of edges inserted incrementally.

ponents do not depend on the size of change in the set of maximal cliques. In `sx-stackoverflow-a2q`, the first two components take around 45% of the time, and in each of `wikipedia-growth`, `youtube-u-growth`, the first two components take around 30% of the time. The significant proportion of time for the first two components means that the overall time for computing the change is not very closely correlated with the size of the change. In the other input graphs, first two components take about 2% of the overall computation time and the change-sensitive behavior can be clearly observed in these plots.

Memory Consumption: Figure 8 shows the main memory used by `IMCE`. For this experiment, we consider two different versions of the algorithm – one with storing the clique set explicitly, and one version with only storing the hashes of the cliques. As expected, the use of a hash function reduces the memory consumption considerably. The difference in memory consumption between the two versions is especially visible in graphs `flickr-growth`, `wiki-talk` and `youtube-u-growth`, where the sizes of the maximal cliques are considerably larger. We used the 64-bit `murmur`[4] hash function on the canonical string representation of a clique, for computing the hash signature. Note that there are some "spikes" in the plot for `dblp-coauthor`, where the memory consumption suddenly increased. On this graph, we observed that the number of maximal cliques at the point corresponding to the spike in memory usage also increased suddenly and then subsequently decreased.

Cumulative Computation time vs. batch size: We also studied the effect of the batch size (ρ) on the cumulative computation time of `IMCE`, while keeping the total number of edges added the same. For example a total of 10,000 edges would lead to 1000 batches if we used a batch size of 10, and 100 batches if we used a batch size of 100. Table 5 shows the results for different batch sizes. There is no observable trend found by varying the batch size.

Summary of Results: To summarize the results of our experiments, we note the following: (1) `IMCE` is change-sensitive: its runtime to enumerate the change in the set of maximal cliques is proportional to the magnitude of the change in the set of maximal cliques. (2) `IMCE` is two to three orders of magnitude faster than prior algorithms (3) the use of hash signatures for storing maximal cliques greatly reduces the memory consumption.

6 Conclusion

We presented change-sensitive algorithms for maintaining the set of maximal cliques in a graph that is changing due to the addition or deletion of edges. We showed nearly tight bounds for the magnitude of change in the set of maximal

cliques, due to a change in the set of edges. Our results show that even for the addition of a small number of edges, the change in the number of maximal cliques can be exponential in the size of the graph, in the worst case. Motivated by this, we designed *change-sensitive* algorithms, whose time complexity of enumerating the change is proportional to the magnitude of the change. Experimental results show that our algorithms are practical and improve on prior work by orders of magnitude.

Many interesting research questions remain open, including: (1) Design of more efficient change-sensitive algorithms for computing $\Lambda^{new}(G, G + H)$, especially for enumerating subsumed cliques. (2) Computation of the the exact value of $\lambda(n)$, the maximum magnitude of change (3) Design of change-sensitive algorithms for other dense structures in a graph such as quasi-cliques.

References

1. 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*, pages 1–25, 2013.
2. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1993.
3. B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *VLDB*, 5(5):454–465, 2012.
4. C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.
5. A. Chateau, P. Riou, and E. Rivals. Approximate common intervals in multiple genome comparison. In *Bioinformatics and Biomedicine (BIBM), 2011 IEEE International Conference on*, pages 131–134. IEEE, 2011.
6. J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *TODS*, 36(4):21, 2011.
7. N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.
8. A. Das and S. Tirthapura. A change-sensitive algorithm for maintaining maximal bicliques in a dynamic bipartite graph. *CoRR*, abs/1707.08272, 2017.
9. 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.
10. D. Duan, Y. Li, R. Li, and Z. Lu. Incremental k-clique clustering in dynamic social networks. *Artificial Intelligence Review*, pages 1–19, 2012.
11. D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC*, pages 403–414, 2010.
12. D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *LNCS*, pages 364–375. 2011.
13. P. Erds and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.
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. `http://faculty.ucr.edu/~hanneman/nettext/`. Textbook on the web.

---

[4] https://sites.google.com/site/murmurhash/

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. M. M.-u. Hussain, A. Wang, and G. Trajcevski. Co-maxrs: Continuous maximizing range-sum query. *Sciences*, 305:110–129, 2015.

18. 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.

19. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

20. I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1):1–30, 2001.

21. F. Kose, W. Weckwerth, T. Linke, and O. Fiehn. Visualizing plant metabolomic correlation networks using clique-metabolite matrices. *Bioinformatics*, 17(12):1198–1208, 2001.

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

23. S. Lehmann, M. Schwartz, and L. K. Hansen. Biclique communities. *Phys. Rev. E*, 78:016108, Jul 2008.

24. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

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

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

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

28. A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu. Densest subgraph in dynamic graph streams. In *MFCS*, pages 472–482, 2015.

29. J. W. Moon and L. Moser. On cliques in graphs. *Israel J. Math.*, 3(1):23–28, 1965.

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

31. A. P. Mukherjee, P. Xu, and S. Tirthapura. Enumeration of maximal cliques from an uncertain graph. *IEEE Trans. Knowl. Data Eng.*, 29(3):543–555, 2017.

32. T. J. Ottosen and J. Vomlel. Honour thy neighbour: clique maintenance in dynamic graphs. In *PGM*, pages 201–208, 2010.

33. 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.

34. 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.

35. 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.

36. 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.

37. V. Stix. Finding all maximal cliques in dynamic graphs. *Comput. Optim. Appl.*, 27(2):173–186, 2004.

38. S. Sun, Y. Wang, W. Liao, and W. Wang. Mining maximal cliques on dynamic graphs efficiently by local strategies. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 115–118. IEEE, 2017.

39. 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.

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

41. 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.

42. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977.

43. 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.

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.